```
function analyse_errors_bins(pos_estimated,score,pos, endbulges)
%analyse_errors_bins(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
if length(pos_estimated) ~= length(score)
   error('pos_estimated  and  score not compatible');
end
if length(pos_estimated) ~= length(pos)
   error('pos_estimated  and  pos not compatible');
end
if length(pos_estimated) ~= length(endbulges)
   error('pos_estimated  and  endbulges size not compatible');
end
N = 100;
Per_bin = 20;
mxscore = max(score);
mnscore = min(score);
dth = (mxscore- mnscore)/N;
thresh = mnscore:dth:mxscore;
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)-Per_bin
   I = find(score >= thresh(i) & score <= thresh(i+Per_bin));
   if ~isempty(I)
      count = count + 1;
      midbin(count) = 0.5*(thresh(i) + thresh(i+Per_bin));
      accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

      J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
      correct_side_dist1(count) = length(J1)/length(I);
      J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
      correct_side_dist2(count) = length(J2)/length(I);
      Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
      correct_side_disth(count) = length(Jh)/length(I);


      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
   else
      count = count+1;
```

```
      midbin(count) = 0.25*(thresh(i) + 2* thresh(i+1) + thresh(i+2));
      accuracy(count) = NaN;
      correct_side_dist1(count) = NaN;
      correct_side_dist2(count) = NaN;
      correct_side_disth(count) = NaN;
      wrong_side(count) = NaN;
      fraction(count) = NaN;
    end
  end

  acc1 = accuracy + correct_side_dist1;
  acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
  clf
  hold on

  plot(midbin, acc2,'g')
  plot(midbin, acc1,'r')
  plot(midbin, accuracy,'b')
  plot(midbin, wrong_side,'k')
  plot(midbin,fraction,'c')

  legend('dist \leq 2', 'dist \leq 1', 'precise', 'wrong side');
  plot(midbin, acc2,'*g')
  plot(midbin, acc1,'or')
  plot(midbin, accuracy,'bd')
  plot(midbin, wrong_side,'kv')
  xlabel('bin');
  %keyboard
  returnfunction analyse_errors_bins1(pos_estimated,score,pos, endbulges,N)
%analyse_errors_bins1(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
if length(pos_estimated) ~= length(score)
    error('pos_estimated  and  score not compatible');
end
if length(pos_estimated) ~= length(pos)
    error('pos_estimated  and  pos not compatible');
end
if length(pos_estimated) ~= length(endbulges)
    error('pos_estimated  and  endbulges size not compatible');
end
if nargin == 4
   N = 6;
end
perc = [1:-1/N:0]*100;
thresh = prctile(score, perc);
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
```

```
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)-1
   I = find(score <= thresh(i) & score >= thresh(i+1));
   if ~isempty(I)
      count = count + 1;
      midbin(count) = mean(score(I));
      accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

      J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
      correct_side_dist1(count) = length(J1)/length(I);
      J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
      correct_side_dist2(count) = length(J2)/length(I);
      Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
      correct_side_disth(count) = length(Jh)/length(I);


      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
   else
      count = count+1;
      midbin(count) = NaN;;
      accuracy(count) = NaN;
      correct_side_dist1(count) = NaN;
      correct_side_dist2(count) = NaN;
      correct_side_disth(count) = NaN;
      wrong_side(count) = NaN;
      fraction(count) = NaN;
   end
end

acc1 = accuracy + correct_side_dist1;
acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
clf
hold on

plot(midbin, acc2,'g')
plot(midbin, acc1,'r')
plot(midbin, accuracy,'b')
plot(midbin, wrong_side,'k')
plot(midbin,fraction,'c')

legend('dist \leq 2', 'dist \leq 1', 'precise', 'wrong side');
plot(midbin, acc2,'*g')
```

```
 plot(midbin, acc1,'or')
 plot(midbin, accuracy,'bd')
 plot(midbin, wrong_side,'kv')
 xlabel('bin');
 %keyboard
 returnfunction analyse_errors_perc(pos_estimated,score,pos, endbulges)
%analyse_errors_perc(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
N = 100;
perc = [1:-1/N:0]*100;
thresh = prctile(score, perc);
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)
   I = find(score > thresh(i));
   if ~isempty(I)
      count = count + 1;
      accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

      J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
      correct_side_dist1(count) = length(J1)/length(I);
      J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
      correct_side_dist2(count) = length(J2)/length(I);
      Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
      correct_side_disth(count) = length(Jh)/length(I);


      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
   else
      count = count+1;
      accuracy(count) = NaN;
      correct_side_dist1(count) = NaN;
      correct_side_dist2(count) = NaN;
      correct_side_disth(count) = NaN;
      wrong_side(count) = NaN;
      fraction(count) = NaN;
   end
 end
```

```
 acc1 = accuracy + correct_side_dist1;
 acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
 clf
 hold on

 plot(perc, acc2,'g')
 plot(perc, acc1,'r')
 plot(perc, accuracy,'b')
 plot(perc, wrong_side,'k')
 plot(perc, thresh,'c')
 legend('dist \leq 2', 'dist \leq 1', 'precise', 'wrong side', 'threshold');
 xlabel('percentage');
 axis([0 100 0 1]);
 %keyboard
 returnfunction analyse_errors_thresh(pos_estimated,score,pos, endbulges)
%analyse_errors_thresh(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
if max(score) > 1
   mxscore = max(score);
else
   mxscore = 1;
end
if min(score) < 0
   mnscore = min(score);
else
   mnscore = 0;
end
Np = 500;
dth = (mxscore- mnscore)/Np;
thresh = mnscore:dth:mxscore;
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
   eb = find(endbulges{i});
   correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)
    I = find(score > thresh(i));
    if ~isempty(I)
       count = count + 1;
       accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

       J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
```

```
    correct_side_dist1(count) = length(J1)/length(I);
    J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
    correct_side_dist2(count) = length(J2)/length(I);
    Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
    correct_side_disth(count) = length(Jh)/length(I);


    wrong_side(count) = sum(1-correct_side(I))/length(I);

    fraction(count) = length(I)/N;
  else
    count = count+1;
    accuracy(count) = NaN;
    correct_side_dist1(count) = NaN;
    correct_side_dist2(count) = NaN;
    correct_side_disth(count) = NaN;
    wrong_side(count) = NaN;
    fraction(count) = NaN;
  end
end

acc1 = accuracy + correct_side_dist1;
acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
clf
hold on

plot(thresh, acc2,'g')
plot(thresh, acc1,'r')
plot(thresh, accuracy,'b')
plot(thresh, wrong_side,'k')
plot(thresh, fraction,'c')
legend('dist \leq 2', 'dist \leq 1', 'precise', 'wrong side', 'fraction');
xlabel('threshold');
%keyboard
returnfunction y = edit_distance(s,t)
% y = edit_distance(s,t)
%  compute edit (levenstein) distance between s and t
C = 0.5; % parameter that fixes the relative
%Algorithm
%
%Construct a matrix containing 0..m rows and 0..n columns.
% Initialize the first row to 0..n.
% Initialize the first column to 0..m.
% 3. Examine each character of s (i from 1 to n).
% 4. Examine each character of t (j from 1 to m).
% 5. If s[i] equals t[j], the cost is 0.
%6. If s[i] doesn't equal t[j], the cost is 1.
% Set cell d[i,j] of the matrix equal to the minimum of:
%a. The cell immediately above plus 1: d[i-1,j] + 1.
%b. The cell immediately to the left plus 1: d[i,j-1] + 1.
```

```
%c. The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost.
%7 After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell d[n,m
n = length(s);
m = length(t);
if n == 0
 y = m;
 return;
end
if m == 0
 y = n;
 return;
end
d = zeros(n+1,m+1); %Construct a matrix containing 0..m rows and 0..n columns.
d(1,:) = [0:m];       % Initialize the first row to 0..n.
d(:,1) = [0:n]';      %Initialize the first column to 0..m.
for i = 1:n
 for j = 1:m
  cost = (s(i) ~= t(j));
  d(i+1,j+1) = min([d(i+1,j)+1, d(i,j+1)+1 , d(i,j)+cost]);
 end
end
y = d(n+1,m+1);
return

function [pos,score] = edit_predict(seqsd, seqs, endbulges)
% y = edit_predict(seqsd, seqs, endbulges)
% find the best matching dicer position by its edit distance to one of the existing dicers
%
% GD 20.2
global Min_dlength  Alpha Step
paramfile = 'edit_params';
%addpath('d:/matlab'); % whereabouts of edit_distance
disp('calculating...');
Step = 1;
fid = fopen(paramfile,'r');
while ~feof(fid)
   line = fgetl(fid);
   eval(line)
end
fclose(fid);
for i = 1:length(seqs)
   %disp(num2str(i));
   [posi, scorei] = edit_predict1(seqsd,seqs{i}, endbulges{i});
   pos(i) = posi;
   score(i) = scorei;
end
return
function [pos, score] = edit_predict1(seqsd,seqsi, endbulgesi);
%calculate the best matching position of dicer
global Min_dlength Alpha Step
```

```
seq_size = length(seqsi);
lb = find(endbulgesi);
eb_size = length(lb);
eb_begin = lb(1);
eb_end = lb(eb_size);
nd = length(seqsd); % number of known dicers
length_seqi = length(seqsi);
%initialize variables with the largest possible distance
min_d = ones(length_seqi,1)*Min_dlength;
mean_d = ones(length_seqi,1)*Min_dlength;
%upper side
  for i = eb_begin-Min_dlength:-Step:1
  p = seqsi(i:i+Min_dlength-1);
  for j = 1:length(seqsd)
    % d(j) = edit_distance(p,seqsd{j});
    d(j) = editD(p,seqsd{j});
%     d(j) = editD(p,seqsd{j});
  end
  min_d(i) = min(d);
  % take also the mean of highest percentile
  [ds,l] = sort(d);
  mean_d(i) = mean(ds(1:floor(Alpha*nd)));
end
for i = eb_end+1:Step:length(seqsi)-Min_dlength+1
  p = seqsi(i:i+Min_dlength-1);
  for j = 1:length(seqsd)
    d(j) = editD(p,seqsd{j});
  end
  min_d(i) = min(d);
  % take also the mean of highest ten percentile
  [ds,l] = sort(d);
  mean_d(i) = mean(ds(1:floor(Alpha*nd)));
end
mmn = min(min_d);
l = find(min_d == mmn);
if length(l) ==1
  pos = l;
  score = Min_dlength - mmn;
else
  % take the position with hte highest alpha score
  [mn,J] = min(mean_d(l));
  pos = l(J);
  score = Min_dlength - mmn;
end
return
function [pos,score] = edit_predictk(seqsd, seqs, endbulges, k, thresh)
% y = edit_predictk(seqsd, seqs, endbulges, k, thresh);
% find the best matching dicer position by its edit distance to one of the existing dicers
% criterion is mean among best k matches
%thresh is the
```

```
% GD 20.2
global Min_dlength Step
paramfile = 'edit_params';
addpath('d:/matlab'); % whereabouts of edit_distance
if nargin <= 4
  thresh = 1.1;
end
if length(seqs) ~= length(endbulges)
  error('size of seqs and endbulges not campatible');
end
if thresh < 1
  error('thresh must be < 1');
end
Step = 1;
fid = fopen(paramfile,'r');
while ~feof(fid)
  line = fgetl(fid);
  eval(line)
end
fclose(fid);
for i = 1:length(seqs)
  disp(num2str(i));
  [posi, scorei] = edit_predict1(seqsd,seqs{i}, endbulges{i},k,thresh);
  pos(i) = posi;
  score(i) = scorei;
end
return
function [pos, score] = edit_predict1(seqsd,seqsi, endbulgesi,k,thresh);
%calculate the best matching position of dicer
global Min_dlength Step
seq_size = length(seqsi);
lb = find(endbulgesi);
eb_size = length(lb);
eb_begin = lb(1);
eb_end = lb(eb_size);
nd = length(seqsd); % number of known dicers
length_seqi = length(seqsi);
%initialize variables with the largest possible distance
min_d = ones(length_seqi,1)*Min_dlength;
mean_d = ones(length_seqi,1)*Min_dlength;
%upper side
for i = eb_begin-Min_dlength:-Step:1
  p = seqsi(i:i+Min_dlength-1);
  for j = 1:length(seqsd)
    d(j) = edit_distance(p,seqsd{j});
  end
  % take also the mean of best k
  [ds,l] = sort(d);
  mean_d(i) = mean(ds(1:k));
end
```

```
%lower side
for i = eb_end+1:Step:length(seqsi)-Min_dlength+1
   p = seqsi(i:i+Min_dlength-1);
   for j = 1:length(seqsd)
      d(j) = edit_distance(p,seqsd{j});
   end
   [ds,I] = sort(d);
   mean_d(i) = mean(ds(1:k));
end
mmn = min(mean_d);
I = find(mean_d <= thresh* mmn);
if length(I) ==1
   pos = I;
   score = Min_dlength - mmn;
else
   % take the position closest to loop
   side = sign(I - eb_begin);
   loopdist = 0.5*(1-side).* (eb_begin - I - Min_dlength) + 0.5*(1+side).* (I- eb_end-1);
   [mndist,J] = min(loopdist);
   I = I(J);
   pos = I;
   score = Min_dlength - mean_d(I);
end
return
function [si,sj] = find_identical_pairs(seqs)
%[si,sj] = find_identical_pairs(seqs)
% find identical palindromes in list
L = length(seqs);
for i = 1:L
   lenp(i) = length(seqs{i});
end
[lenps, I] = sort(lenp);
seqs = seqs(I);
count = 0;
for i = 1:L
   for j = i+1:L
      if lenps(i) ~= lenps(j)
         break
      else
         if all(seqs{i} == seqs{j})
            count = count+1;
            si(count) = I(i);
            sj(count) = I(j);
         end
      end
   end
end

      function strseq = int2nuc(intseq, ncase)
%strseq = int2nuc(intseq, ncase)
```

```
%convert a sequence of '1 2 3 4' into  'A C T G'  or 'a c t g'
% ncase = uppercase | lowercase
if nargin == 1
   ncase = 'uppercase';
end
if strcmp(ncase,'uppercase')
   nucs = 'ACTG';
elseif strcmp(ncase,'lowercase')
   nucs = 'actg';
end
strseq = char(size(intseq));
for i = 1:length(intseq)
   strseq(i) = nucs(intseq(i));
end
return
method = 'poly3'
if method == 'poly2'
% poly2 combined
    %configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
    %points on side error line
    as = [-1.5000   -0.9988   -0.5012   -0.0035    0.5012    0.99    1.4927];
    bs = [0.2325    0.2295    0.1360    0.0804    0.0336    0.0102    0.0015];
%points on precise within2 line
    ap = [-1.4965   -0.9988   -0.5012    0.0035    0.5012    0.9988];
    bp = [0.6798    0.6974    0.8348    0.9196    0.9722    0.9985];
elseif method == 'poly3'
    % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
    % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
     %points on side error line
    as = [-1.4981   -1.2412   -0.9994   -0.5006    0.0019    0.5044    0.7727    0.9994    1.2714    1.5057];
    bs = [0.2303    0.2248    0.2028    0.1239    0.0872    0.0560    0.0211    0.0211         0         0];
    %points on precise within2 line
    ap = [ -1.4981   -1.3281   -1.0031   -0.4931    0.0019    0.4969    0.9994    1.5000];
    bp = [0.6940    0.7000    0.7312    0.8688    0.9165    0.9404    0.9752    1.0000
yside = 1-interp1(as,bs,xi,'linear','extrap')
yprec = interp1(ap,bp,xi,'linear','extrap')
function [yside, yprec2] = interpolate_prob_new(score, fitfile);
%[yside, yprec2] = interpolate_prob_new(score, fitfile);
% load the parameters for interpolation
fid = fopen(fitfile,'r');
while ~feof(fid)
   line = fgetl(fid);
   if ~isstr(line), break , end;
   eval(line)
end
fclose(fid);
%interpolate
yside = interp1(xs,ys,score,'linear');
yprec2 = interp1(xp2,yp2,score,'linear');
returnfunction [yside, yprec2] = interpolate_prob_old_ver5(xi, method);
```

```
%[yside, yprec2] = interpolate_prob_old_ver5(yi, method);
% parameters are configuration specific see below!
disp('these are the accumulated performance, not the actual performance per bin');
disp(' press enter to continue');
pause
if nargin == 1
   method = 'poly3';
end
if strcmp(method, 'poly2')
% poly2 combined
   %configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   %points on side error line
   as = [-1.5000   -0.9988   -0.5012   -0.0035    0.5012    0.99    1.4927];
   bs = [0.2325    0.2295    0.1360    0.0804    0.0336    0.0102    0.0015];
%points on precise within2 line
   ap = [-1.4965   -0.9988   -0.5012    0.0035    0.5012    0.9988];
   bp = [0.6798    0.6974    0.8348    0.9196    0.9722    0.9985];
elseif strcmp(method,'poly3')
   % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
    %points on side error line
   as = [-1.4981   -1.2412   -0.9994   -0.5006    0.0019    0.5044    0.7727    0.9994    1.2714    1.5057];
   bs = [0.2303    0.2248    0.2028    0.1239    0.0872    0.0560    0.0211    0.0211         0         0];
   %points on precise within2 line
   ap = [ -1.4981   -1.3281   -1.0031   -0.4931    0.0019    0.4969    0.9994    1.5000];
   bp = [0.6940    0.7000    0.7312    0.8688    0.9165    0.9404    0.9752    1.0000];
end
yside = 1-interp1(as,bs,xi,'linear');
yprec2 = interp1(ap,bp,xi,'linear');
function [yside, yprec2] = interpolate_probabilities(xi, method);
%[yside, yprec2] = interpolate_probabilities(yi, method);
% parameters are configuration specific see below!
if nargin == 1
   method = 'poly3';
end
if strcmp(method,'poly3')
   % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
    %points on side error line
 as = [1.5000  1.3235  1.2591  0.9478  0.2052  -0.1707  -0.3337  -0.5573  -0.7706  -1.0873];
 bs = [1.0000  1.0000  0.9355  0.8710  0.8485  0.8438  0.8182  0.5806  0.5000  0.5000];
 ap = as;
 bp = [1.0000  1.0000  0.9355  0.8710  0.8485  0.8438  0.7576  0.4839  0.2803  0.2681];
end
yside = 1-interp1(as,bs,xi,'linear','extrap');
yprec2 = interp1(ap,bp,xi,'linear','extrap');
function [yside, yprec2] = interpolate_probabilities_ver5(xi, method);
%[yside, yprec2] = interpolate_probabilities_ver5(yi, method);
% parameters are configuration specific see below!
disp('these are the accumulated performance, not the actual performance per bin');
```

```
disp(' press enter to continue');
pause
if nargin == 1
   method = 'poly3';
end
if strcmp(method, 'poly2')
% poly2 combined
   %configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   %points on side error line
   as = [-1.5000   -0.9988   -0.5012   -0.0035    0.5012    0.99    1.4927];
   bs = [0.2325    0.2295    0.1360    0.0804    0.0336    0.0102    0.0015];
%points on precise within2 line
   ap = [-1.4965   -0.9988   -0.5012    0.0035    0.5012    0.9988];
   bp = [0.6798    0.6974    0.8348    0.9196    0.9722    0.9985];
elseif strcmp(method,'poly3')
   % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
    %points on side error line
   as = [-1.4981   -1.2412   -0.9994   -0.5006    0.0019    0.5044    0.7727    0.9994    1.2714    1.5057];
   bs = [0.2303    0.2248    0.2028    0.1239    0.0872    0.0560    0.0211    0.0211         0         0];
   %points on precise within2 line
   ap = [ -1.4981   -1.3281   -1.0031   -0.4931    0.0019    0.4969    0.9994    1.5000];
   bp = [0.6940    0.7000    0.7312    0.8688    0.9165    0.9404    0.9752    1.0000];
end
yside = 1-interp1(as,bs,xi,'linear');
yprec2 = interp1(ap,bp,xi,'linear');
function [yside, yprec2] = interpolate_probabilities_ver5(xi, method);
%[yside, yprec2] = interpolate_probabilities_ver5(yi, method);
% parameters are configuration specific see below!
disp('these are the accumulated performance, not the actual performance per bin');
disp(' press enter to continue');
pause
if nargin == 1
   method = 'poly3';
end
if strcmp(method, 'poly2')
% poly2 combined
   %configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   %points on side error line
   as = [-1.5000   -0.9988   -0.5012   -0.0035    0.5012    0.99    1.4927];
   bs = [0.2325    0.2295    0.1360    0.0804    0.0336    0.0102    0.0015];
%points on precise within2 line
   ap = [-1.4965   -0.9988   -0.5012    0.0035    0.5012    0.9988];
   bp = [0.6798    0.6974    0.8348    0.9196    0.9722    0.9985];
elseif strcmp(method,'poly3')
   % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
   % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
    %points on side error line
   as = [-1.4981   -1.2412   -0.9994   -0.5006    0.0019    0.5044    0.7727    0.9994    1.2714    1.5057];
   bs = [0.2303    0.2248    0.2028    0.1239    0.0872    0.0560    0.0211    0.0211         0         0];
```

```
    %points on precise within2 line
    ap = [ -1.4981   -1.3281   -1.0031   -0.4931    0.0019    0.4969    0.9994    1.5000];
    bp = [0.6940    0.7000    0.7312    0.8688    0.9165    0.9404    0.9752    1.0000];
end
yside = 1-interp1(as,bs,xi,'linear');
yprec2 = interp1(ap,bp,xi,'linear');
function [yside, yprec2] = interpolate_probabilities_ver5(xi, method);
%[yside, yprec2] = interpolate_probabilities_ver5(yi, method);
% parameters are configuration specific see below!
disp('version 5 does not allow for extrapolation')
if nargin == 1
    method = 'poly3';
end
if strcmp(method,'poly3')
    % configuration: 5'    -2 7 -2 6     3'     -11 0 -11 0
    % alpha5 = 0.9; alpha3 = 0.40; alpha_dlen = 0.2;
     %points on side error line
 as = [1.5000  1.3235  1.2591  0.9478  0.2052  -0.1707  -0.3337  -0.5573  -0.7706  -1.0873];
 bs = [1.0000  1.0000  0.9355  0.8710  0.8485  0.8438  0.8182  0.5806  0.5000  0.5000];
 ap = as;
 bp = [1.0000  1.0000  0.9355  0.8710  0.8485  0.8438  0.7576  0.4839  0.2803  0.2681];
end
%yside = 1-interp1(as,bs,xi,'linear','extrap');
%yprec2 = interp1(ap,bp,xi,'linear','extrap');
yside = 1-interp1(as,bs,xi,'linear');
yprec2 = interp1(ap,bp,xi,'linear');
function len = length_seq(seqs);
%len = length_seq(seqs);
%calculate sequence length
for i = 1:length(seqs)
   len(i) = length(seqs{i});
end
returnl = find(lenp-pos >= 22);
for i = 1:length(l)
frstl(i) = seqs{l(i)}(pos(l(i)));
lastl(i,:) = seqs{l(i)}([20+pos(l(i)), 21+pos(l(i))]);
end %load training data
randomize = 1;
curdir = pwd;
cd d:/rosetta/data_new
load matlab_147_unique.mat
if randomize
 disp('performing randomized permutation');
 l = randperm(length(seqs));
 bulges1 = bulges1(l);
 bulges2 = bulges2(l);
 endbulges = endbulges(l);
 lend = lend(l);
 lenp = lenp(l);
 pos = pos(l);
```

```
 seq_id = seq_id(l);
 seqs = seqs(l);
   seqsd = seqsd(l);
end
cd(curdir) %load training data
randomize = 0;
curdir = pwd;
cd d:/rosetta/data_new
load matlab_147_unique.mat
if randomize
 disp('performing randomized permutation');
 l = randperm(length(seqs));
 bulges1 = bulges1(l);
 bulges2 = bulges2(l);
 endbulges = endbulges(l);
 lend = lend(l);
 lenp = lenp(l);
 pos = pos(l);
 seq_id = seq_id(l);
 seqs = seqs(l);
   seqsd = seqsd(l);
end
cd(curdir) %load training data
curdir = pwd;
cd d:/rosetta/data_new
load matlab_173_unique.mat
cd(curdir) function pos = locate_dicer(dicer_seq,pal_seq);
%pos = locate_dicer(dicer_seq,palseq)
%get absolute position of dicer on palindrom, from the beginning of the pllindrom
if length(dicer_seq) ~= length(pal_seq)
   error('different number of sequences');
end
pos = zeros(1,length(dicer_seq));
for i = 1:length(dicer_seq)
   l = findstr(dicer_seq{i}, pal_seq{i});
   if length(l) == 1
      pos(i) = l;
   else
      pos(i) = NaN;
   end
end
function pos_dummy = make_pos_dummy(seqs,bulges1,bulges2,endbulges)
%pos_dummy = make_pos_dummy(seqs,bulges1,bulges2,endbulges)
% construct dummy pos vector for testing classifiers
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
mode = 'testing'
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
pos_dummy = zeros(1, length(seqs));
for i = 1:length(seqs)
```

```
  pos_dummy(i) = mkpsi0(seqs{i},bulges1{i},bulges2{i},endbulges{i});
  %pos_dummy(i) = mkpsi1(seqs{i},bulges1{i},bulges2{i},endbulges{i});
end
return
% version 0
function posi = mkpsi0(seqsi,bulges1i,bulges2i,endbulgesi)
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
% simple rule
% assume dicer of length 17 exactly.
% nearest , in euclidean distance to some prototype, regardles of distance
% from loop and side
% params are assumed -2 3 -2 7
prototype = [0.3381,-0.4804,0.1813,-0.1205,0.3318,0.0028,0.2095,-0.3635, ...
  -0.0711,-0.1954,-0.3103,-0.3066,0.1822,-0.1972,0.0417,0.3385,-0.4882, ...
  -0.3491,0.1979,-0.1216,0.3600,0.3537,0.0936,0.2271,-0.1907,0.3939, ...
  0.3385,0.0681,-0.1296,0.2027,0.0466,0.2948,0.4568,0.0226,0.0182,...
  0.0828,-0.0765,0.0155,-0.1660,-0.0671,-0.2741,0.0798,-0.3252,0.0678 ...
  0.2604,0.0298,0.1405,-0.2909,-0.1202,0.2833,0.1808,-0.4104,-0.0389];
seq_size = length(seqsi);
lb = find(endbulgesi);
eb_size = length(lb);
eb_begin = lb(1);
[xi, yi] = preprocess5(seqsi,bulges1i,bulges2i,endbulgesi);
[m,n] = size(xi);
sim = xi(:,3:n)*(prototype(1:n-2))';
[maxs,m] = max(sim);
side = xi(m,1);
loopdist = xi(m,2) * (0.5* (seq_size - eb_size));
posi = (1+side)/2*(eb_end + loopdist) + (1-side)/2*(eb_begin - loopdist);
return
% version 1
function posi = mkpsi1(seqsi,bulges1i,bulges2i,endbulgesi)
% simple rule
% assume dicer of length 17 exactly.
% nearest position to loop, such that dicer begins with t , not on bulge1
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
lb = find(endbulgesi);
eb_size = length(lb);
eb_begin = lb(1);
eb_end = lb(eb_size);
pos = find(seqsi == 3 & endbulgesi == 0 & bulges1i == 0);
dst = zeros(size(pos));
if ~isempty(pos)
  side = sign(pos-eb_begin);
  lup = find(side == -1);
  dst(lup) = eb_begin - (pos(lup) + Min_dlength -1);
  ldwn = find(side == 1);
  dst(ldwn) = pos(ldwn) - eb_end;
```

```
   dst(find(dst < 0))= 1000;
   [md,l] = min(dst);
   posi = pos(l);
else
 pos = find(seqsi == 4 & endbulgesi == 0 & bulges1i == 0);
 side = sign(pos-eb_begin);
   lup = find(side == -1);
   dst(lup) = eb_begin - (pos(lup) + Min_dlength -1);
   ldwn = find(side == 1);
   dst(ldwn) = pos(ldwn) - eb_end;
   on_endbulge = find(dst < 0);
   dst(on_endbulge) == 1000;
   [md,l] = min(dst);
   posi = pos(l);
end
return
function [x,y,seqno] = merge_sets(x1,x2,y1,y2,seqno1,seqno2)
%[x,y,seqno] = merge_sets(x1,x2,y1,y2,seqno1,seqno2)
% concatenate datasets
x = [x1; x2];
y = [y1; y2];
seqno = [seqno1; seqno2+max(seqno1)];
return%mfold_cv
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
pos5 = zeros(0);
score5 =zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];% test set
   bt = setdiff(bins_all , bs);% train set

   disp(' ');
   disp([ 'm = ' num2str(m)]);

   [pos5m,score5m] = edit_predict(seqsd(bt), seqs(bs), endbulges(bs));

  pos5 = [pos5,pos5m];
  score5 =[score5,score5m];

  m = m+1;
end
% perform m fold cross validation on article + zuker results by splitting set
validation = 1; % otherwise, only testing is performed
mfold = 5;
n_all = 278;
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
```

```
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
pos3 = zeros(0);
score3 = zeros(0);
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
pos5 = zeros(0);
score5 = zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
% test set
   filename3 = ['svm_tst_3m.dat'];
   filename5 = ['svm_tst_5m.dat'];
   [x3s, seqno3s] = preprocess_and_write_data3(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename3);
   [x5s, seqno5s] = preprocess_and_write_data5(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename5);
   disp([ 'm = ' num2str(m)]);
   disp('written preprocessed test examples');

   bt = setdiff(bins_all , bs);
   filename3 = ['svm_trn_3m.dat'];
   filename5 = ['svm_trn_5m.dat'];
   [x3t, seqno3t] = preprocess_and_write_data3(seqs_all(bt),bulges1_all(bt),bulges2_all(bt),endbulges_all(bt),
filename3, pos_all(bt)+lend_all(bt)-1);
   [x5t, seqno5t] = preprocess_and_write_data5(seqs_all(bt),bulges1_all(bt),bulges2_all(bt),endbulges_all(bt),
filename5, pos_all(bt));
   disp('written preprocessed training examples');

   disp('now train and test svm. results should be in   g:\research\rosetta\svm_light_utils1\svm_outputs\out3m.out,
out5m.out');
   disp(' ***********************************************************************************************************');

   pause
   cd svm_outputs
   load out3m.out
   load out5m.out
   cd ..
   [pos3m, score3m] = svm_position(x3s,out3m,seqno3s, endbulges_all(bs), lenp_all(bs)+lend_all(bs));
   [pos5m, score5m] = svm_position(x5s,out5m,seqno5s, endbulges_all(bs), lenp_all(bs));
   %collect global variables
   x3 = [x3; x3s];
   out3 =[out3;out3m];
   if m == 1
      seqno3 = seqno3s;
   else
      mx3 = max(seqno3);
```

```
        seqno3 = [seqno3 ; mx3+seqno3s];
    end
    pos3 = [pos3 pos3m];
    score3 = [score3 score3m];

    x5 = [x5;x5s];;
    out5 =[out5;out5m];
    if m == 1
        seqno5 = seqno5s;
    else
        mx5 = max(seqno5);
        seqno5 = [seqno5 ; mx3+seqno5s];
    end
    pos5 = [pos5 pos5m];
    score5 = [score5 score5m];
    m = m+1;
end
%mfold_cv_transduction
% perform m fold cross validation on article + zuker results by splitting set
% use transduction mode of SVM
mfold = 5;
n_all = 278;
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
pos3 = zeros(0);
score3 = zeros(0);
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
pos5 = zeros(0);
score5 = zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
   %training set
   bt = setdiff(bins_all , bs);
   filename3 = ['svm_trn_3t.dat'];
   filename5 = ['svm_trn_5t.dat'];
   [x3t, seqno3t] = preprocess_and_write_data3(seqs_all(bt),bulges1_all(bt),bulges2_all(bt),endbulges_all(bt),
filename3, pos_all(bt)+lend_all(bt)-1);
   [x5t, seqno5t] = preprocess_and_write_data5(seqs_all(bt),bulges1_all(bt),bulges2_all(bt),endbulges_all(bt),
filename5, pos_all(bt));
   disp('written preprocessed training examples');

% test set - append to previous file
    [x3s, seqno3s] = preprocess_and_write_data3_tr(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename3);
```

```
    [x5s, seqno5s] = preprocess_and_write_data5_tr(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename5);

    % test set - write to seperate file
    filename3 = ['svm_tst_3t.dat'];
    filename5 = ['svm_tst_5t.dat'];
    [x3s, seqno3s] = preprocess_and_write_data3(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename3);
    [x5s, seqno5s] = preprocess_and_write_data5(seqs_all(bs),bulges1_all(bs),bulges2_all(bs),endbulges_all(bs),
filename5);
  disp('written preprocessed testing examples');
    disp([ 'm = ' num2str(m)]);
    disp('written preprocessed test examples');

disp('now train and test svm. ');
disp('transductive data are in  svm_trn_5t.dat  etc.')
disp('results should be in   g:\research\rosetta\svm_light_utils1\svm_outputs\out5t.out, etc.');
disp(' ***************************************************************************************************************');

    pause
    cd svm_outputs
    load out3t.out
    load out5t.out
    cd ..
    [pos3t, score3t] = svm_position(x3s,out3t,seqno3s, endbulges_all(bs), lenp_all(bs)+lend_all(bs));
    [pos5t, score5t] = svm_position(x5s,out5t,seqno5s, endbulges_all(bs), lenp_all(bs));
    %collect global variables
    x3 = [x3; x3s];
    out3 =[out3;out3t];
    if m == 1
        seqno3 = seqno3s;
    else
        mx3 = max(seqno3);
        seqno3 = [seqno3 ; mx3+seqno3s];
    end
    pos3 = [pos3 pos3t];
    score3 = [score3 score3t];

    %here am


    x5 = [x5;x5s];;
    out5 =[out5;out5t];
    if m == 1
        seqno5 = seqno5s;
    else
        mx5 = max(seqno5);
        seqno5 = [seqno5 ; mx3+seqno5s];
    end
    pos5 = [pos5 pos5t];
```

```
    score5 = [score5 score5t];
    m = m+1;
end
% perform m fold cross validation on article + zuker results by splitting set
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename3 = 'd:/svm_light/model3m';
tst_filename3 = 'd:/rosetta/svm_light_utils1/svm_tst_3m.dat';
trn_filename3 = 'd:/rosetta/svm_light_utils1/svm_trn_3m.dat';
out_filename3 = 'd:/rosetta/svm_light_utils1/out3m.out';
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
m = 1;
while m <= mfold
    bs = [bins(m)+1: bins(m+1)];
% test set

[x3s, seqno3s] = preprocess_and_write_data3(seqs(bs),bulges1(bs), ...
   bulges2(bs),endbulges(bs), tst_filename3);
    disp([ 'm = ' num2str(m)]);
    disp('written preprocessed test examples');

    bt = setdiff(bins_all , bs);

    [x3t, seqno3t] = preprocess_and_write_data3(seqs(bt),bulges1(bt),...
       bulges2(bt),endbulges(bt), trn_filename3, pos(bt)+lend(bt)-1);
    disp('written preprocessed training examples');

   dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename3 ' ' model_filename3]);
    dos(['d:/svm_light/svm_classify ' tst_filename3 ' ' model_filename3 ' ' out_filename3]);

    load out3m.out

    %collect global variables
    x3 = [x3;x3s];
    out3 =[out3;out3m];
    if m == 1
        seqno3 = seqno3s;
    else
        mx3 = max(seqno3);
        seqno3 = [seqno3 ; mx3+seqno3s];
    end
    m = m+1;
end
clear x3s x3t out3m seqno3s seqno3t bs bt
% just for printing  the info
```

```
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
disp(' ');
disp('***********************************************************');
disp('3 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);
% perform m fold cross validation on article + zuker results by splitting set
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename3 = 'd:/svm_light/model3m';
tst_filename3 = 'd:/rosetta/svm_light_utils1/svm_tst_3mb.dat';
trn_filename3 = 'd:/rosetta/svm_light_utils1/svm_trn_3mb.dat';
out_filename3 = 'd:/rosetta/svm_light_utils1/out3mb.out';
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
% test set

[x3s, seqno3s] = preprocess_and_write_data3(seqs(bs),bulges1(bs), ...
  bulges2(bs),endbulges(bs), tst_filename3);
   disp([ 'm = ' num2str(m)]);
   disp('written preprocessed test examples');

   bt = setdiff(bins_all , bs);

   [x3t, seqno3t] = preprocess_and_write_data3(seqs(bt),bulges1(bt),...
      bulges2(bt),endbulges(bt), trn_filename3, pos(bt)+lend(bt)-1);
   disp('written preprocessed training examples');

  dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename3 ' ' model_filename3]);
   dos(['d:/svm_light/svm_classify ' tst_filename3 ' ' model_filename3 ' ' out_filename3]);

   load out3mb.out
   out3m= out3mb;

   %collect global variables
   x3 = [x3;x3s];;
   out3 =[out3;out3m];
   if m == 1
      seqno3 = seqno3s;
   else
      mx3 = max(seqno3);
      seqno3 = [seqno3 ; mx3+seqno3s];
   end
```

```
    m = m+1;
end
clear x3s x3t out3m out3mb seqno3s seqno3t bs bt
% just for printing  the info
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
disp(' ');
disp('*******************************************************');
disp('3 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);
% perform m fold cross validation on article + zuker results by splitting set
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename5 = 'd:/svm_light/model5m';
tst_filename5 = 'd:/rosetta/svm_light_utils1/svm_tst_5m.dat';
trn_filename5 = 'd:/rosetta/svm_light_utils1/svm_trn_5m.dat';
out_filename5 = 'd:/rosetta/svm_light_utils1/out5m.out';
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
m = 1;
while m <= mfold
    bs = [bins(m)+1: bins(m+1)];
% test set

[x5s, seqno5s] = preprocess_and_write_data5(seqs(bs),bulges1(bs), ...
   bulges2(bs),endbulges(bs), tst_filename5);
    disp([ 'm = ' num2str(m)]);
    disp('written preprocessed test examples');

    bt = setdiff(bins_all , bs);

    [x5t, seqno5t] = preprocess_and_write_data5(seqs(bt),bulges1(bt),...
       bulges2(bt),endbulges(bt), trn_filename5, pos(bt));
    disp('written preprocessed training examples');

   dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename5 ' ' model_filename5]);
    dos(['d:/svm_light/svm_classify ' tst_filename5 ' ' model_filename5 ' ' out_filename5]);

    load out5m.out

    %collect global variables
    x5 = [x5;x5s];;
    out5 =[out5;out5m];
    if m == 1
        seqno5 = seqno5s;
    else
```

```
        mx5 = max(seqno5);
        seqno5 = [seqno5 ; mx5+seqno5s];
    end
    m = m+1;
end
clear x5s x5t out5m seqno5s seqno5t bs bt
% just for printing  the info
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
disp(' ');
disp('*******************************************************');
disp('5 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);
% perform m fold cross validation on article + zuker results by splitting set
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename5 = 'd:/svm_light/model5m';
tst_filename5 = 'd:/rosetta/svm_light_utils1/svm_tst_5m.dat';
trn_filename5 = 'd:/rosetta/svm_light_utils1/svm_trn_5m.dat';
out_filename5 = 'd:/rosetta/svm_light_utils1/out5m.out';
model_filename3 = 'd:/svm_light/model3m';
tst_filename3 = 'd:/rosetta/svm_light_utils1/svm_tst_3m.dat';
trn_filename3 = 'd:/rosetta/svm_light_utils1/svm_trn_3m.dat';
out_filename3 = 'd:/rosetta/svm_light_utils1/out3m.out';
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
pos3 = zeros(0);
score3 = zeros(0);
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
pos5 = zeros(0);
score5 = zeros(0);
m = 1;
while m <= mfold
    bs = [bins(m)+1: bins(m+1)];
% test set
    [x3s, seqno3s] = preprocess_and_write_data3(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), tst_filename3);
    [x5s, seqno5s] = preprocess_and_write_data5(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), tst_filename5);
    disp([ 'm = ' num2str(m)]);
    disp('written preprocessed test examples');

    bt = setdiff(bins_all , bs);
    [x3t, seqno3t] = preprocess_and_write_data3(seqs(bt),bulges1(bt),bulges2(bt),endbulges(bt), trn_filename3,
pos(bt)+lend(bt)-1);
    [x5t, seqno5t] = preprocess_and_write_data5(seqs(bt),bulges1(bt),bulges2(bt),endbulges(bt), trn_filename5,
pos(bt));
```

```
    disp('written preprocessed training examples');

    dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename5 ' ' model_filename5]);
    dos(['d:/svm_light/svm_classify ' tst_filename5 ' ' model_filename5 ' ' out_filename5]);
     dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename3 ' ' model_filename3]);
    dos(['d:/svm_light/svm_classify ' tst_filename3 ' ' model_filename3 ' ' out_filename3]);

    load(out_filename3);
    load(out_filename5)

    %[pos3m, score3m] = svm_position(x3s,out3m,seqno3s, endbulges(bs), lenp(bs));
    %[pos5m, score5m] = svm_position(x5s,out5m,seqno5s, endbulges(bs), lenp(bs));
    %collect global variables
    x3 = [x3; x3s];
    out3 =[out3;out3m];
    if m == 1
        seqno3 = seqno3s;
    else
        mx3 = max(seqno3);
        seqno3 = [seqno3 ; mx3+seqno3s];
    end
%    pos3 = [pos3 pos3m];
%    score3 = [score3 score3m];

    x5 = [x5;x5s];;
    out5 =[out5;out5m];
    if m == 1
        seqno5 = seqno5s;
    else
        mx5 = max(seqno5);
        seqno5 = [seqno5 ; mx3+seqno5s];
    end
 %   pos5 = [pos5 pos5m];
 %   score5 = [score5 score5m];
    m = m+1;
end
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
disp(' ');
disp('*********************************************************');
disp('5 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
disp(' ');
disp('*********************************************************');
disp('3 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);% perform m fold cross validation on article + zuker results by splitting set
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
```

```
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename5 = 'd:/svm_light/model5mb';
tst_filename5 = 'd:/rosetta/svm_light_utils1/svm_tst_5mb.dat';
trn_filename5 = 'd:/rosetta/svm_light_utils1/svm_trn_5mb.dat';
out_filename5 = 'd:/rosetta/svm_light_utils1/out5mb.out';
model_filename3 = 'd:/svm_light/model3mb';
tst_filename3 = 'd:/rosetta/svm_light_utils1/svm_tst_3mb.dat';
trn_filename3 = 'd:/rosetta/svm_light_utils1/svm_trn_3mb.dat';
out_filename3 = 'd:/rosetta/svm_light_utils1/out3mb.out';
x3 = zeros(0);
out3 =zeros(0);
seqno3 = zeros(0);
pos3 = zeros(0);
score3 = zeros(0);
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
pos5 = zeros(0);
score5 = zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
% test set
   [x3s, seqno3s] = preprocess_and_write_data3(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), tst_filename3);
   [x5s, seqno5s] = preprocess_and_write_data5(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), tst_filename5);
   disp([ 'm = ' num2str(m)]);
   disp('written preprocessed test examples');

   bt = setdiff(bins_all , bs);
   [x3t, seqno3t] = preprocess_and_write_data3(seqs(bt),bulges1(bt),bulges2(bt),endbulges(bt), trn_filename3,
pos(bt)+lend(bt)-1);
   [x5t, seqno5t] = preprocess_and_write_data5(seqs(bt),bulges1(bt),bulges2(bt),endbulges(bt), trn_filename5,
pos(bt));
   disp('written preprocessed training examples');

   dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename5 ' ' model_filename5]);
   dos(['d:/svm_light/svm_classify ' tst_filename5 ' ' model_filename5 ' ' out_filename5]);
    dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename3 ' ' model_filename3]);
   dos(['d:/svm_light/svm_classify ' tst_filename3 ' ' model_filename3 ' ' out_filename3]);

   load(out_filename3);
   load(out_filename5)
   out5m = out5mb;
   out3m = out3mb;

   %[pos3m, score3m] = svm_position(x3s,out3m,seqno3s, endbulges(bs), lenp(bs));
   %[pos5m, score5m] = svm_position(x5s,out5m,seqno5s, endbulges(bs), lenp(bs));
   %collect global variables
   x3 = [x3; x3s];
```

```
   out3 =[out3;out3m];
   if m == 1
      seqno3 = seqno3s;
   else
      mx3 = max(seqno3);
      seqno3 = [seqno3 ; mx3+seqno3s];
   end
%    pos3 = [pos3 pos3m];
%    score3 = [score3 score3m];

   x5 = [x5;x5s];;
   out5 =[out5;out5m];
   if m == 1
      seqno5 = seqno5s;
   else
      mx5 = max(seqno5);
      seqno5 = [seqno5 ; mx3+seqno5s];
   end
 %   pos5 = [pos5 pos5m];
 %   score5 = [score5 score5m];
   m = m+1;
end
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
disp(' ');
disp('*********************************************************');
disp('5 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
disp(' ');
disp('*********************************************************');
disp('3 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);% perform m fold cross validation on article + zuker results by splitting set
% modified file names for input/output so that can be run in parralel
mfold = 5;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
svm_params = input('enter svm parameters: ','s');
model_filename5 = 'd:/svm_light/model5m_b';
tst_filename5 = 'd:/rosetta/svm_light_utils1/svm_tst_5m_b.dat';
trn_filename5 = 'd:/rosetta/svm_light_utils1/svm_trn_5m_b.dat';
out_filename5 = 'd:/rosetta/svm_light_utils1/out5m_b.out';
x5 = zeros(0);
out5 =zeros(0);
seqno5 = zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
% test set
```

```
   [x5s, seqno5s] = preprocess_and_write_data5(seqs(bs),bulges1(bs), ...
      bulges2(bs),endbulges(bs), tst_filename5);
    disp([ 'm = ' num2str(m)]);
    disp('written preprocessed test examples');

    bt = setdiff(bins_all , bs);

    [x5t, seqno5t] = preprocess_and_write_data5(seqs(bt),bulges1(bt),...
      bulges2(bt),endbulges(bt), trn_filename5, pos(bt));
    disp('written preprocessed training examples');

   dos(['d:/svm_light/svm_learn ' svm_params ' ' trn_filename5 ' ' model_filename5]);
    dos(['d:/svm_light/svm_classify ' tst_filename5 ' ' model_filename5 ' ' out_filename5]);

    load out5m_b.out

    %collect global variables
    x5 = [x5;x5s];;
    out5 =[out5;out5m_b];
    if m == 1
       seqno5 = seqno5s;
    else
       mx5 = max(seqno5);
       seqno5 = [seqno5 ; mx5+seqno5s];
    end
    m = m+1;
end
clear x5s x5t out5m_b seqno5s seqno5t bs bt
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
disp(' ');
disp('*********************************************************');
disp('5 prime end');
disp(['params          : ' num2str([Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength]) ]);
disp(['svm light params: ' svm_params]);
%mfold_cvk
mfold = 8;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
k = 4;
pos5 = zeros(0);
score5 =zeros(0);
m = 1;
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];% test set
   bt = setdiff(bins_all , bs);% train set

   disp([ 'm = ' num2str(m)]);
```

```
   [pos5m,score5m] = edit_predictk(seqsd(bt), seqs(bs), endbulges(bs),k);

  pos5 = [pos5,pos5m];
  score5 =[score5,score5m];

  m = m+1;
end
function [intseq, fault_seq] = nuc2int4_new(strseq);
%[intseq, fault_seq] = nuc2int4_new(strseq)
%convert a sequence of 'A C T G' into a array of 1 2 3 4
intseq = zeros(size(strseq));
fault_seq = 0;
for i = 1:length(strseq)
   switch upper(strseq(i))
      case 'A' , intseq(i) = 1;
      case 'C' , intseq(i) = 2;
      case 'T' , intseq(i) = 3;
      case 'G' , intseq(i) = 4;
      otherwise , intseq = []; fault_seq = 1; break;
   end
end
function run_edit_distance()
infile='c:\editdistance\draw_file.dat';
outfile='c:\editdistance\dicer_res.dat';
cd \\rosetta4\Development\gideon\edit_dist
seqsd = cell(0);
ii=0
fid=fopen('seqsd','r');
while ~feof(fid)
   ii=ii+1;
   seqsd{ii}=fgetl(fid);

end
fclose(fid);
fidin = fopen(infile,'r');
fidout = fopen(outfile,'w');
fidin
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
   disp('reading structure...');
   [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);
   [pos,score] = edit_predict(seqsd, seqs, endbulges)

%write to file
%seq_id0 is added so as to sequential order of sequence numbers
 seq_id = seq_id + seq_id0;
 res = [seq_id; pos; score];
   fprintf(fidout, '%d %d %g ', res);
   seq_id0 = max(seq_id);
```

```
end
fclose(fidin);
fclose(fidout);
quit
return;
function y = prctile(x,p);
%PRCTILE gives the percentiles of the sample in X.
%   Y = PRCTILE(X,P) returns a value that is greater than P percent
%   of the values in X. For example, if P = 50  Y is the median of X.
%
%   P may be either a scalar or a vector. For scalar P, Y is a row
%   vector containing Pth percentile of each column of X. For vector P,
%   the ith row of Y is the P(i) percentile of each column of X.
%   Copyright (c) 1993-98 by The MathWorks, Inc.
%   $Revision: 2.6 $  $Date: 1997/11/29 01:46:27 $
[prows pcols] = size(p);
if prows ~= 1 & pcols ~= 1
    error('P must be a scalar or a vector.');
end
if any(p > 100) | any(p < 0)
    error('P must take values between 0 and 100');
end
xx = sort(x);
[m,n] = size(x);
if m==1 | n==1
    m = max(m,n);
 if m == 1,
    y = x*ones(length(p),1);
    return;
 end
    n = 1;
    q = 100*(0.5:m - 0.5)./m;
    xx = [min(x); xx(:); max(x)];
else
    q = 100*(0.5:m - 0.5)./m;
    xx = [min(x); xx; max(x)];
end
q = [0 q 100];
y = interp1(q,xx,p);
function seqtable = prepare_seqtable(seqno_list);
% seqtable = prepare_seqtable(seqno);
%seqtable conatins for each seqno its starting location in example list
% and its end location
seqtable = zeros(max(seqno_list),2);
i = 1;
seqno = seqno_list(i);
while i <= length(seqno_list)
   seqtable(seqno,1) = i;
   while seqno_list(i) == seqno
      seqtable(seqno,2) = i;
```

```
    i = i+1;
    if i > length(seqno_list)
      break
    end
  end
  if i > length(seqno_list)
      break
  end
  seqno = seqno_list(i);
end
return



  function [x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges, filename, pos)
%[x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges,filename, pos+lend-1);
%[x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges,filename);   %testing mode
% notice that here pos is pos
%x12 are the first two elements of x (side , relative_loopdist)
%high level function for preparing data and writing for svm training
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
if nargin == 5
   mode = 'testing';
else
   mode = 'training';
end
x12 = zeros(0);
seqno = zeros(0);
fid = fopen(filename,'w');
for i = 1:length(seqsp)
  if strcmp(mode,'training')
    [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
  elseif strcmp(mode,'testing')
    [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i},NaN);
  end
  write_examples(xi, yi, fid);
  x12 = [x12; xi(:,1:2)];
  seqno = [seqno; i*ones(size(xi,1),1)];

  if mod(i,100) == 0; i, end
end
fclose(fid);
return
  function [x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges, filename, pos)
%[x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges,filename, pos+lend-1);
%[x12, seqno] = preprocess_and_write_data3(seqsp,bulges1,bulges2,endbulges,filename);   %testing mode
% notice that here pos is pos
%x12 are the first two elements of x (side , relative_loopdist)
```

```
%high level function for preparing data and writing for svm training
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params3.dat');
if nargin == 5
    mode = 'testing';
else
    mode = 'training';
end
x12 = zeros(0);
seqno = zeros(0);
fid = fopen(filename,'a');
for i = 1:length(seqsp)
  if strcmp(mode,'training')
     [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
  elseif strcmp(mode,'testing')
     [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i},NaN);
  end
  write_examples(xi, yi, fid);
  x12 = [x12; xi(:,1:2)];
  seqno = [seqno; i*ones(size(xi,1),1)];

  if mod(i,100) == 0; i, end
end
fclose(fid);
return
  function [x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges, filename, pos)
%[x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges,filename, pos);
%[x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges,filename);   %testing mode
%x12 are the first two elements of x (side , relative_loopdist)
%high level function for preparing data and writing for svm training
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
if nargin == 5
    mode = 'testing';
else
    mode = 'training';
end
%Maxsize is a simple upper bound for the number of possible positions
Maxsize = 0;
for i = 1:length(seqsp);
  Maxsize = Maxsize+length(seqsp{i});
end
x12 = zeros(Maxsize,2);
seqno = zeros(Maxsize,1);
xfrom = 1; % index where to write into xi and seqno
fid = fopen(filename,'w');
for i = 1:length(seqsp)
```

```
  if strcmp(mode,'training')
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
  elseif strcmp(mode,'testing')
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i},NaN);
  end

  write_examples(xi, yi, fid);
  xlength = size(xi,1);

  x12(xfrom: xfrom + xlength-1,:) = xi(:,1:2);
  seqno(xfrom: xfrom + xlength-1) = i*ones(xlength,1);
  xfrom = xfrom + xlength ;

  if mod(i,1000) == 0; disp(i); end
end
fclose(fid);
% remove the unneeded sapce in x12 and seqno
x12(xfrom:Maxsize,:) = [];
seqno(xfrom:Maxsize) = [];
return
function [x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges, filename, pos)
%[x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges,filename, pos);
%[x12, seqno] = preprocess_and_write_data5(seqsp,bulges1,bulges2,endbulges,filename);   %testing mode
%x12 are the first two elements of x (side , relative_loopdist)
%high level function for preparing data and writing for svm training
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
if nargin == 5
   mode = 'testing';
else
   mode = 'training';
end
x12 = zeros(0);
seqno = zeros(0);
fid = fopen(filename,'a');
for i = 1:length(seqsp)
  if strcmp(mode,'training')
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
  elseif strcmp(mode,'testing')
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i},NaN);
  end

  write_examples(xi, yi, fid);
  x12 = [x12; xi(:,1:2)];
  seqno = [seqno; i*ones(size(xi,1),1)];

  if mod(i,100) == 0; i, end
end
fclose(fid);
```

```
return
function [x,y,seqno] = preprocess_data3(seqsp,bulges1,bulges2,endbulges,pos)
%[x,y,seqno] = preprocess_data3(seqsp,bulges1,bulges2,endbulges,pos);   % for training
%[x,y,seqno] = preprocess_data3(seqsp,bulges1,bulges2,endbulges);    % for testing
%
% 3' side of MiR
%
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Mindlength] = read_params('params3.dat');
if nargin == 5
  mode = 'training';
else
  mode = 'testing';
end
x = zeros(0);
y = zeros(0);
seqno = zeros(0);
if strcmp(mode,'training')
  for i = 1:length(seqsp)
    [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
    x = [x; xi];
    y = [y; yi];
    seqno = [seqno; i*ones(size(yi))];
    if mod(i,10) == 0; i, end
  end
else
  for i = 1:length(seqsp)
    [xi, yi] = preprocess3(seqsp{i},bulges1{i},bulges2{i},endbulges{i});
    x = [x; xi];
    y = [y; yi];  % this is just a list of zeros
    seqno = [seqno; i*ones(size(yi))];
    if mod(i,10) == 0; i, end
  end
end
returnfunction [x,y,seqno] = preprocess_data5(seqsp,bulges1,bulges2,endbulges,pos)
%[x,y,seqno] = preprocess_data5(seqsp,bulges1,bulges2,endbulges,pos); % for training
%[x,y,seqno] = preprocess_data5(seqsp,bulges1,bulges2,endbulges); % for testing
%high level function for preparing data for svm training
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
Nnucfrom = -2; % nucleotides region of interest
Nnucto = 7;
Nbfrom = -2; %bulges region of interest
Nbto = 6;
Min_dlength = 17; % min dicer length
if nargin == 5
  mode = 'training';
else
  mode = 'testing';
```

```
end
x = zeros(0);
y = zeros(0);
seqno = zeros(0);
if strcmp(mode,'training')
  for i = 1:length(seqsp)
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i},pos(i));
    x = [x; xi];
    y = [y; yi];
    seqno = [seqno; i*ones(size(yi))];
    if mod(i,10) == 0; i, end
  end
else
  for i = 1:length(seqsp)
    [xi, yi] = preprocess5(seqsp{i},bulges1{i},bulges2{i},endbulges{i});
    x = [x; xi];
    y = [y; yi];  % this is just a list of zeros
    seqno = [seqno; i*ones(size(yi))];
    if mod(i,10) == 0; i, end
  end
end
returnfunction x = preprocess_window(posj, seqwin,bulges1win,bulges2win, seq_size, eb_size, eb_begin, eb_end)
%preprocess_window : lower level function
% produces a feature vector from 3windows of the sequence
global Nnucfrom Nnucto Nbfrom Nbto mode
lenx = 2 + length(seqwin) *4 + 2*length(bulges1win);
x = zeros(0);
side = sign(posj-eb_begin);
x(1) = side; % -1 for upper, 1 for lower
loopdist = (1 + side)/2 * (posj - eb_end) + ...   % lower part
  (1 -side)/2* (eb_begin - posj);            % upper part
% normalize x2 by palyndrom available length
x(2) = loopdist/(0.5* (seq_size - eb_size));
n_assigned = 2;
binseq = zeros(4, Nnucto+1-Nnucfrom);
binseq([0:size(binseq,2)-1]*4 + seqwin) = 1;
binseq = binseq(:)';
x(n_assigned+1: n_assigned +length(binseq))  = binseq;
n_assigned = n_assigned + length(binseq);
x(n_assigned+1: n_assigned +2*length(bulges1win)) = [bulges1win bulges2win];
returnfunction [xi, yi] = preprocess3(seqspi,bulges1i,bulges2i,endbulgesi,posi)
% low level function aimed at processing a sigle sequence
%in testing mode, yi are simply 0;
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
seq_size = length(seqspi); % size of palindrome = # nucleotides
l = find(endbulgesi);
eb_size = length(l); % size of endbulge = loop
eb_begin = l(1);
eb_end = l(eb_size);
```

```
xi = zeros(0);
yi = zeros(0);
% range include for upper and lower 5' positions
from = min(Nbfrom,Nnucfrom);
to = max(Nbto, Nnucto);
for side = -1:2:1
  if side == -1
    posrange = Min_dlength : eb_begin-1-to;
  else
    posrange = eb_end+Min_dlength : seq_size-to;
  end

  for j = 1:length(posrange)
    posj = posrange(j);

    nuc_win = posj+Nnucfrom:posj+Nnucto;  %window of nucleotides (sequence)
    b_win = posj+Nbfrom:posj+Nbto;   %window of bulges (1 sided & 2 sided)


 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if length(seqspi) < max(nuc_win)
      disp('bug1')
    end
    if length(bulges1i) < max(b_win)| length(bulges2i) < max(b_win)
      disp('bug2')
    end
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    xij = preprocess_window(posj, seqspi(nuc_win), ...
        bulges1i(b_win),bulges2i(b_win), seq_size, eb_size, eb_begin, eb_end);
    xi = [xi; xij];
    if strcmp(mode,'training')
      yij = (posj == posi)*2-1; %+1 or -1
      yi = [yi ; yij];
   else
      yi = [yi ; 0];
    end
  end  % for j = 1:length(posrange)
end % if side ==
returnfunction [xi, yi] = preprocess5(seqspi,bulges1i,bulges2i,endbulgesi,posi)
% low level function aimed at processing a sigle sequence
%in testing mode, yi are simply 0;
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
%disp('preprocess5 modified. target id triangle like near 5 prime end');
seq_size = length(seqspi); % size of palindrome = # nucleotides
I = find(endbulgesi);
eb_size = length(I); % size of endbulge = loop
eb_begin = I(1);
eb_end = I(eb_size);
```

```
xi = zeros(0);
yi = zeros(0);
% range include for upper and lower 5' positions
from = min(Nbfrom,Nnucfrom);
to = max(Nbto, Nnucto);
for side = -1:2:1
  if side == -1
    posrange = 1+abs(from) :eb_begin-Min_dlength;
  else
    posrange = eb_end+1+abs(from) : seq_size+1-Min_dlength;
  end

  for j = 1:length(posrange)
    posj = posrange(j);

    nuc_win = posj+Nnucfrom:posj+Nnucto;  %window of nucleotides (sequence)
    b_win = posj+Nbfrom:posj+Nbto;   %window of bulges (1 sided & 2 sided)
    xij = preprocess_window(posj, seqspi(nuc_win), ...
        bulges1i(b_win),bulges2i(b_win), seq_size, eb_size, eb_begin, eb_end);
    xi = [xi; xij];

    if strcmp(mode,'training')
        yij = (posj == posi)*2-1; %+1 or -1
      % new version suitable for regression
    %yij = max(1-0.5*abs(posj-posi), -1);  % giving 1 at max and -1 at distance 3 or more
      yi = [yi ; yij];
    else
      yi = [yi ; 0];
    end
  end  % for j = 1:length(posrange)
end % if side ==
return[pos5,score5] = svm_position(x5,out5, seqno5, endbulges, lenp);
svkernel = input('enter kernel name: ','s');
targetdir = input('enter target directory name (e.g.) params-1-10-1-10: ' ,'s');
targetfile = ['d:\rosetta\svm_light_utils1\figures_174\' targetdir '\' svkernel];
figure(1); analyse_errors_thresh(pos5,score5,pos,endbulges); title(svkernel);
eval(['print -djpeg90 ' targetfile 'thresh']);
figure(2); analyse_errors_perc(pos5,score5,pos,endbulges); title(svkernel);
eval(['print -djpeg90 ' targetfile 'perc']);
in='c:\rosetta\data_baseline_15_5\draw_file2K.dat';
o = 'edist_res_file_hmdc257_2000pals.txt';
run_edit_distance_ranit(in,o);
in='c:\rosetta\data_baseline_15_5\200VirusesDraw.txt';
o = 'edist_res_file_hmdc257_virus.txt';
run_edit_distance_ranit(in,o);
in='c:\rosetta\data_baseline_15_5\badPalsGrade.txt';
o = 'edist_res_file_hmdc257_lowpal.txt';
run_edit_distance_ranit(in,o);
in='c:\rosetta\data_baseline_15_5\goodPalsGrade33.txt';
o = 'edist_res_file_hmdc257_highpal.txt';
```

```
run_edit_distance_ranit(in,o);function [Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params(paramsfile);
%[Nnucfrom, Nnucto, Nbfrom, Nbto, Mindlength] = read_params(paramsfile);
Nfields = 5;
fieldnames = cell(Nfields);
fieldnames(1:Nfields) = {'Nnucfrom'; 'Nnucto'; 'Nbfrom'; 'Nbto' ; 'Min_dlength'};
fid = fopen(paramsfile,'r');
while ~feof(fid)
   line = fgetl(fid);
   [field, rest] = strtok(line);

   if ~isempty(rest)
      value = num2str(strtok(rest));
   else
      error(['value of ' field  '  not specified']);
   end

   % assign the value to the proper variable
   found = 0;
   for i = 1:Nfields
      if strcmp(field, fieldnames{i})
         eval([field '=' num2str(value) ';']);
         found = 1;
         break
      end
   end
   if found == 0
      error(['illegal field  ' field  ]);
   end
end
fclose(fid);
return

function [seqs,len] = read_seq(filename);
%[len,seqs] = read_seq(filename);
%reads dicer or pal sequences into cell array, in numeric format
fid = fopen(filename,'r');
if fid == -1
  error([' file ' filename '  could not be opened']);
end
id = 0;
seq_no = 0;
while ~feof(fid)
  line = fgetl(fid);
  line = deblank(line);
  [intseq, fault_seq] = nuc2int4_new(line);
  id = id + 1;
  if fault_seq == 0
    seq_no = seq_no + 1;
    seqs{seq_no} = intseq;
    len(seq_no) = length(intseq);
```

```
  else
    disp(['faulty seq on id  ' num2str(id)])
  end

  if(mod(seq_no,1000) == 0 & seq_no ~= 0)
     disp(['seq_no ' num2str(seq_no)]);
  end
end
fclose(fid);
return

  function [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure(filename);
%[seqs,bulges1,bulges2,endbulges,seq_id] = read_structure(filename)
% read zuker structure
% seq is a cell array containing sequences
% bulge1 is a cell array with binary strings with 1 for one sided bulge (not incl. end bulge)
% bulge2 is similarly for 2 sided bulge
% endbulge is a cell array with binary strings with 1 on the end bulge only
Mxplen = 250;  % maximal length of palindrom
if nargin == 0
  filename = 'C:\rosetta_versions\ver9\data\zuker_draw_z.txt';
end
fid = fopen(filename,'r');
seq_no = 0;
seqs = cell(0);
bulges1= cell(0);
bulges2= cell(0);
endbulges = cell(0);
seq_id = zeros(0);
id = 0;
while ~feof(fid)
  structure = char(4,250);
  for i = 1:4
    line = fgetl(fid);
    structure(i,1:length(line)) = line;
  end
  id = id +1;
  [seqi, bulge1i, bulge2i, endbulgei] = get_features(structure);
  [intseq, fault_seq] = nuc2int4_new(seqi);
  if fault_seq == 0
      seq_no = seq_no + 1;
      seqs{seq_no} = intseq;
      bulges1{seq_no} = bulge1i;
      bulges2{seq_no} = bulge2i;
      endbulges{seq_no} = endbulgei;
      seq_id(seq_no) = id;
   else
      disp(['faulty seq on id  ' num2str(id)])
   end
  if(mod(seq_no,1000) == 0)
```

```
      seq_no
    end
  end
  fclose(fid);
  return
  function [seq, bulge1, bulge2, endbulge] = get_features(structure)
  % get sequence as well as bulge structure
  %upper half (5' side)
  bulge_row = 1; % the row of bulge letters
  bulge_row_opposite = 4;
  uphalf = structure(1:2,:);
  [j,k] = find(isletter(uphalf));
  max_col = max(k);
  count = 0;
  for col =1: max_col
    fl = find(isletter(uphalf(:,col)));
    if ~isempty(fl)
      count = count + 1;
      seq(count) = uphalf(fl,col);
      bulge = (fl == bulge_row);
      bulge1(count) = 0;
      bulge2(count) = 0;
      if bulge & isletter(structure(bulge_row_opposite,col))
        bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
        bulge1(count) = 1;
      end
    end
  end
  % endbulge is coded on the upper half
  % go backwards form 3' side to 5' side
  endbulge = zeros(size(bulge1));
  pos = length(bulge1);
  while bulge1(pos) == 1
    endbulge(pos) = 1;
    bulge1(pos) = 0;
    pos = pos - 1;
  end
  %lower half
  bulge_row = 2; % 4 th line on structure is 2 line on lower half
  bulge_row_opposite = 1;
  lwhalf = structure(3:4,:);
  [j,k] = find(isletter(lwhalf));
  max_col = max(k);
  for col =max_col:-1:1
    fl = find(isletter(lwhalf(:,col)));
    if ~isempty(fl)
      count = count + 1;
      seq(count) = lwhalf(fl,col);
      bulge = (fl == bulge_row);
```

```
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
return

  function [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fid,seqtot);
%[seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fid,seqtot)
% file id version: read 'seqtot' zuker draw palindromes from file handle 'fid'
%
% read zuker structure
% seq is a cell array containing sequences
% bulge1 is a cell array with binary strings with 1 for one sided bulge (not incl. end bulge)
% bulge2 is similarly for 2 sided bulge
% endbulge is a cell array with binary strings with 1 on the end bulge only
Mxplen = 250;  % maximal length of palindrom
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
[Nnucfrom, Nnucto, Nbfrom, Nbto, Min_dlength] = read_params('params5.dat');
seq_no = 0;
seqs = cell(0);
bulges1= cell(0);
bulges2= cell(0);
endbulges = cell(0);
seq_id = zeros(0);
id = 0;
while ~feof(fid) & seq_no < seqtot
  structure = char(4,250);
  for i = 1:4
    line = fgetl(fid);
    structure(i,1:length(line)) = line;
  end
  id = id +1;
  [seqi, bulge1i, bulge2i, endbulgei] = get_features(structure);
  [intseq, fault_seq] = nuc2int4_new(seqi);
  fault_structure = check_structure(seqi, bulge1i, bulge2i, endbulgei);

  if fault_seq == 0 & fault_structure == 0
      seq_no = seq_no + 1;
      seqs{seq_no} = intseq;
      bulges1{seq_no} = bulge1i;
      bulges2{seq_no} = bulge2i;
      endbulges{seq_no} = endbulgei;
      seq_id(seq_no) = id;
```

```
    else
        disp(['faulty seq on id  ' num2str(id)])
    end
   if(mod(seq_no,1000) == 0)
      seq_no
   end
end
return
function [seq, bulge1, bulge2, endbulge] = get_features(structure)
% get sequence as well as bulge structure
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
count = 0;
for col =1: max_col
   fl = find(isletter(uphalf(:,col)));
   if ~isempty(fl)
      count = count + 1;
      seq(count) = uphalf(fl,col);
      bulge = (fl == bulge_row);
      bulge1(count) = 0;
      bulge2(count) = 0;
      if bulge & isletter(structure(bulge_row_opposite,col))
         bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
         bulge1(count) = 1;
      end
   end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
pos = length(bulge1);
if(pos < 1)
   return
end
while bulge1(pos) == 1
   endbulge(pos) = 1;
   bulge1(pos) = 0;
   pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
lwhalf = structure(3:4,:);
[j,k] = find(isletter(lwhalf));
max_col = max(k);
```

```
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge = (fl == bulge_row);
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
return
function fault_structure = check_structure(seqi, bulge1i, bulge2i, endbulgei)
%test whether structure can be worked out by classifier, e.g.
% length of sequence is too short not enough space for Mir of length Mindlength
global Nnucfrom Nnucto Nbfrom Nbto mode
global Min_dlength
seq_size = length(seqi);
lb = find(endbulgei);
if(isempty(lb))
  fault_structure=1
  return
end
eb_size = length(lb);
eb_begin = lb(1);
eb_end = lb(eb_size);
% how many nucleotides/bulges are taken before 5' position
from = min(Nbfrom,Nnucfrom);
if (1+abs(from) > eb_begin-Min_dlength)  & ...
    (eb_end+1+abs(from) > seq_size+1-Min_dlength)
  fault_structure = 1;
else
  fault_structure = 0;
end
return

  function [seqs,bulges1,bulges2,endbulges,seq_id, conn] = read_structure_new(filename);
%[seqs,bulges1,bulges2,endbulges,seq_id,conn] = read_structure_new(filename)
% read zuker structure
% updated 22.1
% extractes also connection structure:
% conn(i) = index of nucleotide connected to nucleotide i (0 if unconnected);
% seq is a cell array containing sequences
% bulge1 is a cell array with binary strings with 1 for one sided bulge (not incl. end bulge)
% bulge2 is similarly for 2 sided bulge
```

```
% endbulge is a cell array with binary strings with 1 on the end bulge only
Mxplen = 250;  % maximal length of palindrom
if nargin == 0
   filename = 'C:\rosetta_versions\ver9\data\zuker_draw_z.txt';
end
fid = fopen(filename,'r');
seq_no = 0;
seqs = cell(0);
bulges1= cell(0);
bulges2= cell(0);
endbulges = cell(0);
seq_id = zeros(0);
conn = cell(0);
id = 0;
while ~feof(fid)
   structure = char(4,250);
   for i = 1:4
      line = fgetl(fid);
      structure(i,1:length(line)) = line;
   end
   id = id +1;
   [seqi, bulge1i, bulge2i, endbulgei, conni] = get_features(structure);
   [intseq, fault_seq] = nuc2int4_new(seqi);
   if fault_seq == 0
        seq_no = seq_no + 1;
        seqs{seq_no} = intseq;
        bulges1{seq_no} = bulge1i;
        bulges2{seq_no} = bulge2i;
        endbulges{seq_no} = endbulgei;
        seq_id(seq_no) = id;
        conn{seq_no} = conni;
    else
        disp(['faulty seq on id  ' num2str(id)])
    end
   if(mod(seq_no,1000) == 0)
      seq_no
   end
end
return
function [seq, bulge1, bulge2, endbulge, conn] = get_features(structure)
% get sequence as well as bulge structure
% sequence index of nucleotide in structure
structure_seq_ind = zeros(size(structure));
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
[j,k] = find(isletter(structure(1:2,:)));
max_col = max(k);
count = 0;
for col =1: max_col
```

```
  fl = find(isletter(structure(1:2,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = structure(1:2,col);
    bulge = (fl == bulge_row);
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    structure_seq_ind(col, fl) = count;
  end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
pos = length(bulge1);
while bulge1(pos) == 1
  endbulge(pos) = 1;
  bulge1(pos) = 0;
  pos = pos - 1;
end
%lower half
bulge_row = 4; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
[j,k] = find(isletter(structure(3:4,:)));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(structure(3:4,col)));
  if ~isempty(fl)
    fl = fl+2; % add 2 since fl = 1/2 on structure(3:4,:
    count = count + 1;
    seq(count) = structure(fl,col);
    bulge = (fl == bulge_row);
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
    structure_seq_ind(col, fl) = count;
  end
end
% produce connection structure
conn = zeros(size(seq));
[j,k] = find(structure_seq_ind(2:3,:) ~= 0);
```

```
j_opp = 5-j; %opposite to j.    3 <-> 2
% produce connection matrix in simple representation
for i = 1:length(j)
  conn(structure_seq_ind(j(i),k(i))) = structure_seq_ind(j_opp(i),k(i));
end
return
function [seqsd, seqs,bulges1,bulges2,endbulges,seq_id] = remove_duplicates(seqsd,
seqs,bulges1,bulges2,endbulges,seq_id);
%[seqsd, seqs,bulges1,bulges2,endbulges,seq_id] = remove_duplicates(seqsd,
seqs,bulges1,bulges2,endbulges,seq_id);
% locate only unique palindrome-dicer pairs
% dicers must be sorted lexicographically
if length(seqsd) ~= length(seqs)
  error('seqsd  and seqs not compatible');
end
Idl = zeros(length(seqsd),1); %entries to be deleted
for i = 2:length(seqsd)
  if length(seqsd{i}) == length(seqsd{i-1}) & length(seqs{i}) == length(seqs{i-1})
    if all(seqsd{i} == seqsd{i-1}) & all(seqs{i} == seqs{i-1})
      Idl(i) = 1;
    end
  end
end
%delete duplicates
I = find(Idl);
seqsd(I) = [];
seqs(I) = [];
bulges1(I) = [];
bulges2(I) = [];
endbulges(I) = [];
seq_id(I) = [];
return

% perform a partitioned testing on large data
mfold = 3;
n_all = length(seqs);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
m = 1;
fname = 'res_poly3_33156.out';
fid = fopen(fname, 'a');
while m <= mfold
   bs = [bins(m)+1: bins(m+1)];
% test set
   filename3 = ['svm3_33156m.dat'];
   filename5 = ['svm5_33156m.dat'];

   [x3s, seqno3s] = preprocess_and_write_data3(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), filename3);
   [x5s, seqno5s] = preprocess_and_write_data5(seqs(bs),bulges1(bs),bulges2(bs),endbulges(bs), filename5);
   disp([ 'm = ' num2str(m)]);
```

```
disp('written preprocessed test examples');

disp('now run svm_classify. inputs are svm3_33156m.dat and svm5_33156m.dat');
disp('results should be in   g:\research\rosetta\svm_light_utils1\svm_outputs\out3m.out, out5m.out');
disp(' ***************************************************************************************************************');

pause
cd svm_outputs

% test that both out files exist
files_ok = 0;
while files_ok == 0;
   files_ok = 1;
   fid5 = fopen('out5m.out','r');
   fid3 = fopen('out3m.out','r');

   if fid5 == -1
      files_ok = 0;
      disp('run svm_classify on 3 data.  out3m.out not found. enter when ready');
      pause
   else
      fclose(fid5);
   end

   if fid3 == -1
      files_ok = 0;
      disp('run svm_classify on 3 data.  out3m.out not found. enter when ready');
      pause
   else
      fclose(fid3);
   end
end

load out3m.out
load out5m.out
%delete files to insure that on next iteration, files are new
delete out3m.out
delete out5m.out

cd ..

%[pos3m, score3m] = svm_position(x3s,out3m,seqno3s, endbulges(bs), lenp(bs));
%[pos5m, score5m] = svm_position(x5s,out5m,seqno5s, endbulges(bs), lenp(bs));
[pos53m, score53m] = svm_position53(x5s,out5m,seqno5s, x3s,out3m,seqno3s, endbulges(bs), lenp(bs));


[yside, yprec2] = interpolate_probabilities(score53m, 'poly3');

res =  [seq_id(bs); pos53m(:,1)'; pos53m(:,2)'; score53m'; yside'; yprec2'];
fprintf(fid, '%d %d %d %g %g %g\n', res);
```

```
    m = m+1;
end
fclose(fid);
function run_edit_distance()
%run_edit_distance(dicerfile, palfile, outfile)
fitfile = '\\rosetta4\Development\gideon\edit_dist\fit_21_025_1.txt'; %suitable for parameter alpha = 0.25
dicerfile='\\rosetta4\Development\gideon\edit_dist\seqsd'
palfile='c:\editdistance\draw_file.dat';
outfile='c:\editdistance\dicer_res.dat';
cd \\rosetta4\Development\gideon\edit_dist
[seqsd,len] = read_seq(dicerfile);
%transform to string
length(seqsd)
for i = 1: length(seqsd)
  seqsd{i} = int2nuc(seqsd{i},'uppercase');
end
fidin = fopen(palfile,'r');
fidout = fopen(outfile,'a');
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
  disp('reading structure...');
  [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);

  %transform back to string
  for i = 1: length(seqs)
    seqs{i} = int2nuc(seqs{i},'uppercase');
  end

  [pos,score] = edit_predict(seqsd, seqs, endbulges)

  %write to file
  %seq_id0 is added so as to sequential order of sequence numbers
  seq_id = seq_id + seq_id0;

  % interpolate
  [yside, yprec2] = interpolate_prob_new(score, fitfile);
  res = [seq_id; pos; score; yprec2;yside];
  fprintf(fidout, '%d %d %g %g %g ', res);
  seq_id0 = max(seq_id);
end
fclose(fidin);
fclose(fidout);
quit
function run_edit_distance()
infile='c:\editdistance\draw_file.dat';
outfile='c:\editdistance\dicer_res.dat';
cd \\rosetta4\Development\gideon\edit_dist
seqsd = cell(0);
```

```
ii=0
fid=fopen('seqsd','r');
while ~feof(fid)
  ii=ii+1;
  seqsd{ii}=fgetl(fid);

end
fclose(fid);
fidin = fopen(infile,'r');
fidout = fopen(outfile,'w');
fidin
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
  disp('reading structure...');
  [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);
  [pos,score] = edit_predict(seqsd, seqs, endbulges)

%write to file
%seq_id0 is added so as to sequential order of sequence numbers
 seq_id = seq_id + seq_id0;
 res = [seq_id; pos; score];
  fprintf(fidout, '%d %d %g ', res);
  seq_id0 = max(seq_id);
end
fclose(fidin);
fclose(fidout);
quit
return;
function run_edit_distance_ranit(palfile,outfile)
fitfile = 'fit_21_025_1.txt'; %suitable for parameter alpha = 0.25
dicerfile='seqsd_hmdc257';
[seqsd,len] = read_seq(dicerfile);
%transform to string
length(seqsd)
for i = 1: length(seqsd)
  seqsd{i} = int2nuc(seqsd{i},'uppercase');
end
fidin = fopen(palfile,'r');
fidout = fopen(outfile,'w');
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
  disp('reading structure...');
  [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);

  %transform back to string
  for i = 1: length(seqs)
    seqs{i} = int2nuc(seqs{i},'uppercase');
  end
```

```
  [pos,score] = edit_predict(seqsd, seqs, endbulges)

  %write to file
  %seq_id0 is added so as to sequential order of sequence numbers
  seq_id = seq_id + seq_id0;

  % interpolate
  [yside, yprec2] = interpolate_prob_new(score, fitfile);
  res = [seq_id; pos; score; yprec2;yside];
  fprintf(fidout, '%d %d %g %g %g\n', res);
  seq_id0 = max(seq_id);
end
fclose(fidin);
fclose(fidout);
function [pos, score] = svm_position(x,svm_score,seqno, endbulges, lenp);
%[pos, score] = svm_position(x,svm_score, seqno, endbulges, lenp);
% postprocess svm outputs (for error analysis)
if size(x,1) ~= size(svm_score,1)
   error('x  and svm_score not compatible');
end
if size(x,1) ~= size(seqno,1)
   error('x  and seqno not compatible');
end
if max(seqno) ~= length(endbulges)
   error('seqno entries and endbulges size not compatible');
end
if length(lenp) ~= length(endbulges)
   error('lenp entries and endbulges size not compatible');
end
% use seqno to produce a list of boundaries  between examples of different sequences
% this is important for efficiency (O(n) instead of O(n^2 log n)
ds = diff(seqno);
bnd = find(ds);
boundaries = [0 bnd' length(seqno)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
for s = 1:max(seqno)
   l = boundaries(s)+1 : boundaries(s+1);
   [maxs,m] = max(svm_score(l));
   score(s) = maxs;

   seq_size = lenp(s);
   lb = find(endbulges{s});
   eb_size = length(lb);
   eb_begin = lb(1);
   eb_end = lb(eb_size);

   side = x(l(m),1);
   loopdist = x(l(m),2) * (0.5* (seq_size - eb_size));
   pos(s) = (1+side)/2*(eb_end + loopdist) + (1-side)/2*(eb_begin - loopdist);
end
```

```
pos = round(pos);
returnfunction [pos, score] = svm_position_r(x,svm_score,seqno, endbulges, lenp);
%[pos, score] = svm_position_r(x,svm_score, seqno, endbulges, lenp);
% postprocess svm outputs (for error analysis)
% regression version
if size(x,1) ~= size(svm_score,1)
   error('x  and svm_score not compatible');
end
if size(x,1) ~= size(seqno,1)
   error('x  and seqno not compatible');
end
if max(seqno) ~= length(endbulges)
   error('seqno entries and endbulges size not compatible');
end
if length(lenp) ~= length(endbulges)
   error('lenp entries and endbulges size not compatible');
end
% use seqno to produce a list of boundaries  between examples of different sequences
% this is important for efficiency (O(n) instead of O(n^2 log n)
ds = diff(seqno);
bnd = find(ds);
boundaries = [0 bnd' length(seqno)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
w = [-1.0  -0.5  0.0  0.5  1.0  0.5  0.0  -0.5  -1.0]; % window for convolution
nws = 0.5*(length(w)-1);
for s = 1:max(seqno)
  I = boundaries(s)+1 : boundaries(s+1);
  svm_scoreI = svm_score(I);
  cnv = conv(w,svm_scoreI);
  lcnv = length(cnv);
  % delete nws values on either side of cnv so that size equals that of scoreI
  cnv([1:nws , lcnv-nws+1:lcnv]) = [];
   [maxs,m] = max(cnv);
   score(s) = maxs;

   seq_size = lenp(s);
   lb = find(endbulges{s});
   eb_size = length(lb);
   eb_begin = lb(1);
   eb_end = lb(eb_size);

   side = x(I(m),1);
   loopdist = x(I(m),2) * (0.5* (seq_size - eb_size));
   pos(s) = (1+side)/2*(eb_end + loopdist) + (1-side)/2*(eb_begin - loopdist);
end
pos = round(pos);
returnfunction [pos, score] = svm_position_soft(x,svm_score,seqno, endbulges, lenp);
%[pos, score] = svm_position_soft(x,svm_score, seqno, endbulges, lenp);
% postprocess svm outputs (for error analysis)
%
% takes the position closest to loop from positions which are at least
```

```
%  best score - (1-Thresh)*abs(best score)
%
Thresh = 0.8;
if size(x,1) ~= size(svm_score,1)
   error('x  and svm_score not compatible');
end
if size(x,1) ~= size(seqno,1)
   error('x  and seqno not compatible');
end
if max(seqno) ~= length(endbulges)
   error('seqno entries and endbulges size not compatible');
end
if length(lenp) ~= length(endbulges)
   error('lenp entries and endbulges size not compatible');
end
% use seqno to produce a list of boundaries  between examples of different sequences
% this is important for efficiency (O(n) instead of O(n^2 log n)
ds = diff(seqno);
bnd = find(ds);
boundaries = [0 bnd' length(seqno)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
for s = 1:max(seqno)
   seq_size = lenp(s);
   lb = find(endbulges{s});
   eb_size = length(lb);
   eb_begin = lb(1);
   eb_end = lb(eb_size);

   l = boundaries(s)+1 : boundaries(s+1);

   maxs = max(svm_score(l));
   score(s) = maxs;
   m = find(svm_score(l) >= maxs - (1-Thresh)*abs(maxs));
   loopdist = x(l(m),2) * (0.5* (seq_size - eb_size));
   minlpdst = min(loopdist);
   lminloopdist = find(loopdist == minlpdst);
   m = m(lminloopdist);
   if length(m > 1)
      mscore = svm_score(l(m));
      [mxs,i] = max(mscore);
      m = m(i);
   end

   side = x(l(m),1);
   loopdist = x(l(m),2) * (0.5* (seq_size - eb_size));
   pos(s) = (1+side)/2*(eb_end + loopdist) + (1-side)/2*(eb_begin - loopdist);
end
pos = round(pos);
returnfunction [pos, score] = svm_position53(x5,svm_score5, seqno5, x3, svm_score3, seqno3, endbulges, lenp);
%[pos, score] = svm_position53(x5,svm_score5, seqno5, x3, svm_score3, seqno3, endbulges, lenp);
% postprocess svm outputs (for error analysis)
```

```
global Maxpos
method = 'bestn';
param = 1;
Maxpos = 10; % maximal number of positions returned
alpha5 = 0.6; alpha3 = 0.40; alpha_dlen = 0.4; % relative weights of 5 and 3 predictions
disp(['alpha5  alpha3  alpha_dlen'  num2str(alpha5) ' ' ...
     num2str(alpha3) ' ' num2str(alpha_dlen)]);
if size(x5,1) ~= size(svm_score5,1)
   error('x5  and svm_score5 not compatible');
end
if size(x3,1) ~= size(svm_score3,1)
   error('x3  and svm_score3 not compatible');
end
if size(x5,1) ~= size(seqno5,1)
   error('x5  and seqno5 not compatible');
end
if size(x3,1) ~= size(seqno3,1)
   error('x3  and seqno3 not compatible');
end
if max(seqno5) ~= max(seqno3)
   error('seqno5 and seqno3 not compatible');
end
if max(seqno5) ~= length(endbulges)
   error('seqno entries and endbulges size not compatible');
end
if length(lenp) ~= length(endbulges)
   error('lenp entries and endbulges size not compatible');
end
fid = fopen('d:\rosetta\svm_light_utils1\dicer_length.out','r');
dlen = str2num(fgetl(fid));
pdlen = str2num(fgetl(fid));
fclose(fid);
dlenmin = min(dlen);
dlenmax = max(dlen);
scdlen = log(pdlen); % scale to score - heuristic!!!
scdlen = scdlen + mean(scdlen);
nseq = max(seqno5);
% use seqno to produce a list of boundaries  between examples of different sequences
% this is important for efficiency (O(n) instead of O(n^2 log n)
ds5 = diff(seqno5);
bnd5 = find(ds5);
boundaries5 = [0 bnd5' length(seqno5)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
ds3 = diff(seqno3);
bnd3 = find(ds3);
boundaries3 = [0 bnd3' length(seqno3)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
if strcmp(method, 'bestn')
   pos = zeros(nseq,2*param);
   score = zeros(nseq,param);
elseif strcmp(method, 'best plus other side')
    pos = zeros(nseq,2*2);
```

```
    score = zeros(nseq,2);
else
    error('not supported yest');
end
for s = 1:max(seqno5)
    l5 = boundaries5(s)+1   : boundaries5(s+1);
    l3 = boundaries3(s)+1 : boundaries3(s+1);

    score5 = svm_score5(l5);
    score3 = svm_score3(l3);

    seq_size = lenp(s);
    lb = find(endbulges{s});
    eb_size = length(lb);
    eb_begin = lb(1);
    eb_end = lb(eb_size);

    side5 = x5(l5,1);
    loopdist5 = x5(l5,2) * (0.5* (seq_size - eb_size));
    pos5 = (1+side5)/2.*(eb_end + loopdist5) + (1-side5)/2.*(eb_begin - loopdist5);
    pos5 = round(pos5);

    side3 = x3(l3,1);
    loopdist3 = x3(l3,2) * (0.5* (seq_size - eb_size));
    pos3 = (1+side3)/2.*(eb_end + loopdist3) + (1-side3)/2.*(eb_begin - loopdist3);
    pos3 = round(pos3);
    % initialize.  pos53(:,1) contains 5' position , pos53(:,1) contains 3' position
    pos53 = zeros(length(l5)*size(pdlen,2),2);
    score53 = zeros(length(l5)*size(pdlen,2),1);
    count = 0;
    for i = 1:length(pos5);
        pos5i = pos5(i);
        J = find(pos3 >= pos5i + dlenmin -1 & pos3 <= pos5i + dlenmax -1 & side3 == side5(i));
        for j = 1:length(J);
            count = count+1;
            pos53(count,:) = [pos5i , pos3(J(j))];
            ind =  pos3(J(j))-pos5i -dlenmin +2;
            if ind < 1 | ind > size(scdlen,2)
                disp('error')
            end

            score53(count) = alpha5*score5(i) + alpha3*score3(J(j)) + alpha_dlen*scdlen(ind);
            %score53(count) = alpha5*tanh(score5(i)) + alpha3*tanh(score3(J(j))) + alpha_dlen*scdlen(ind);
            %score53(count) = max(score5(i),score3(J(j))) + alpha_dlen*scdlen(ind);
        end
    end
    % now pick the desired positions for each sequence ,
    % e.g. 'best' , 'best plus other side', 'above thresh' 'percentile'.
    l = find(pos53(:,1) == 0);
    pos53(l,:) = [];
```

```
    score53(l) = [];
    if isempty(score53)
        error('empty score53');
    end

    [poss, scores] = choose_pos_score(pos53,score53, eb_begin, method, param);
    pos(s,:) = poss;
    score(s,:) = scores;
    if mod(s,1000) == 0
        disp([num2str(s)])
    end
end
return
function [pos, score] = choose_pos_score(pos53,score53, eb_begin, method, param)
% auxilary function
if strcmp(method, 'bestn')
    nbest = param;
    [s,I] = sort(-score53);
    pos(1:nbest,:) = pos53(I(1:nbest),:);
    score(1:nbest) = score53(I(1:nbest));
    pos = pos';
    pos = pos(:);
    pos = pos';
    score = score';
elseif strcmp(method, 'best plus other side')
    pos = zeros(2,2);
    score = zeros(1,2);

    [mx,i] = max(score53);
    pos(1,:) = pos53(i,:);
    score(1) = score53(i);

    Os = find( (pos53(:,1)-eb_begin) * (pos(1,1) -eb_begin)  < 0); %Other side
    if ~isempty(Os)
        [mx,i] = max(score53(Os));
        pos(2,:) = pos53(Os(i),:);
        score(2) = score53(Os(i));
    else
        % this may hapen when the sequence on other side was too short.
        pos(2,:) = NaN;
        score(2) = NaN;
    end
elseif strcmp(method, 'percentile')
    perc = params;
    if perc < 1
        perc = perc*100;
    end
    xp = prctile(score53, perc);
    I = find(score53 >= xp);
    [s,J] = sort(-score53(I));
```

```
   J = I(J);
   score = score53(I);
   pos = pos53(I,:);
else
   error('method not implemented');
end
returnfunction [pos, score] = svm_position53h(x5,svm_score5, seqno5, x3, svm_score3, seqno3, endbulges, lenp);
%[pos, score] = svm_position53h(x5,svm_score5, seqno5, x3, svm_score3, seqno3, endbulges, lenp);
% postprocess svm outputs (for error analysis)
% hard limiter on results of classifier on 3'
global Maxpos
method = 'bestn';
param = 1;
Maxpos = 10; % maximal number of positions returned
alpha5 = 0.6; alpha3 = 0.40; alpha_dlen = 0.4; % relative weights of 5 and 3 predictions
disp(['alpha5  alpha3  alpha_dlen'  num2str(alpha5) ' ' ...
     num2str(alpha3) ' ' num2str(alpha_dlen)]);
if size(x5,1) ~= size(svm_score5,1)
   error('x5  and svm_score5 not compatible');
end
if size(x3,1) ~= size(svm_score3,1)
   error('x3  and svm_score3 not compatible');
end
if size(x5,1) ~= size(seqno5,1)
   error('x5  and seqno5 not compatible');
end
if size(x3,1) ~= size(seqno3,1)
   error('x3  and seqno3 not compatible');
end
if max(seqno5) ~= max(seqno3)
   error('seqno5 and seqno3 not compatible');
end
if max(seqno5) ~= length(endbulges)
   error('seqno entries and endbulges size not compatible');
end
if length(lenp) ~= length(endbulges)
   error('lenp entries and endbulges size not compatible');
end
fid = fopen('d:\rosetta\svm_light_utils1\dicer_length.out','r');
dlen = str2num(fgetl(fid));
pdlen = str2num(fgetl(fid));
fclose(fid);
dlenmin = min(dlen);
dlenmax = max(dlen);
scdlen = log(pdlen); % scale to score - heuristic!!!
scdlen = scdlen + mean(scdlen);
nseq = max(seqno5);
% use seqno to produce a list of boundaries  between examples of different sequences
% this is important for efficiency (O(n) instead of O(n^2 log n)
ds5 = diff(seqno5);
```

```
bnd5 = find(ds5);
boundaries5 = [0 bnd5' length(seqno5)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
ds3 = diff(seqno3);
bnd3 = find(ds3);
boundaries3 = [0 bnd3' length(seqno3)];  % examples of sequence i are between boundaries(i)+1 and boundaries(i+1)
if strcmp(method, 'bestn')
   pos = zeros(nseq,param);
   score = zeros(nseq,param);
elseif strcmp(method, 'best plus other side')
    pos = zeros(nseq,2);
   score = zeros(nseq,2);
else
   error('not supported yest');
end
for s = 1:max(seqno5)
   l5 = boundaries5(s)+1   : boundaries5(s+1);
   l3 = boundaries3(s)+1 : boundaries3(s+1);

   score5 = svm_score5(l5);
   score3 = svm_score3(l3);

   seq_size = lenp(s);
   lb = find(endbulges{s});
   eb_size = length(lb);
   eb_begin = lb(1);
   eb_end = lb(eb_size);

   side5 = x5(l5,1);
   loopdist5 = x5(l5,2) * (0.5* (seq_size - eb_size));
   pos5 = (1+side5)/2.*(eb_end + loopdist5) + (1-side5)/2.*(eb_begin - loopdist5);
   pos5 = round(pos5);

   if max(score3 < -0.1)
     [maxs,m] = max(svm_score5);
     score(s) = maxs;
     pos(s) = po5(m);
   else
     side3 = x3(l3,1);
     loopdist3 = x3(l3,2) * (0.5* (seq_size - eb_size));
     pos3 = (1+side3)/2.*(eb_end + loopdist3) + (1-side3)/2.*(eb_begin - loopdist3);
     pos3 = round(pos3);
   % initialize.  pos53(:,1) contains 5' position , pos53(:,1) contains 3' position
    pos53 = zeros(length(l5)*size(pdlen,2),1);
    score53 = zeros(length(l5)*size(pdlen,2),1);
    count = 0;
    for i = 1:length(pos5);
       pos5i = pos5(i);
       J = find(pos3 >= pos5i + dlenmin -1 & pos3 <= pos5i + dlenmax -1 & side3 == side5(i));
       for j = 1:length(J);
          count = count+1;
```

```
            pos53(count,:) = pos5i;
            ind =  pos3(J(j))-pos5i -dlenmin +2;
            if ind < 1 | ind > size(scdlen,2)
               disp('error')
            end

            score53(count) = alpha5*score5(i) + alpha3*score3(J(j)) + alpha_dlen*scdlen(ind);
         end
      end
      % now pick the desired positions for each sequence ,
      % e.g. 'best' , 'best plus other side', 'above thresh' 'percentile'.
      I = find(pos53(:,1) == 0);
      pos53(I) = [];
      score53(I) = [];
      if isempty(score53)
         error('empty score53');
      end

      [poss, scores] = choose_pos_score(pos53,score53, eb_begin, method, param);
      pos(s,:) = poss;
      score(s,:) = scores;
      if mod(s,1000) == 0
         disp([num2str(s)])
      end
   end
end
return
function [pos, score] = choose_pos_score(pos53,score53, eb_begin, method, param)
% auxilary function
if strcmp(method, 'bestn')
   nbest = param;
   [s,I] = sort(-score53);
   pos(1:nbest) = pos53(I(1:nbest));
   score(1:nbest) = score53(I(1:nbest));
   pos = pos';
   score = score';
elseif strcmp(method, 'best plus other side')
   pos = zeros(1,2);
   score = zeros(1,2);

   [mx,i] = max(score53);
   pos(1) = pos53(i);
   score(1) = score53(i);

   Os = find( (pos53(:,1)-eb_begin) * (pos(1,1) -eb_begin)  < 0); %Other side
   if ~isempty(Os)
      [mx,i] = max(score53(Os));
      pos(2) = pos53(Os(i));
      score(2) = score53(Os(i));
   else
```

```
        % this may hapen when the sequence on other side was too short.
        pos(2) = NaN;
        score(2) = NaN;
    end
elseif strcmp(method, 'percentile')
    perc = params;
    if perc < 1
        perc = perc*100;
    end
    xp = prctile(score53, perc);
    I = find(score53 >= xp);
    [s,J] = sort(-score53(I));
    J = I(J);
    score = score53(I);
    pos = pos53(I  );
else
    error('method not implemented');
end
returnfunction svm_predict(infile,outfile);
%svm_predict(infile,outfile);
%perform svm position prediction
svm_light_folder = 'C:/svm/bin/';
model_filename5 = [svm_light_folder 'model5-2-6-2-6-21'];
tst_filename5 = 'C:/svm/Temp/svm_tst_5.dat';
svm_out_filename5 = 'C:/svm/Temp/out5.out';
fit_filename = 'C:/svm/Score/fit_p5-2-6-2-6-21.txt';
[seqs,bulges1,bulges2,endbulges,seq_id] = read_structure(infile);

[x5, seqno5] = preprocess_and_write_data5(seqs,bulges1, ...
   bulges2,endbulges, tst_filename5);
dos([svm_light_folder 'svm_classify ' tst_filename5 ' ' model_filename5 ' ' svm_out_filename5]);
% load and postprocess
curdir=pwd;
cd 'c:/svm/temp'
load out5.out;
cd(curdir);
lenp = length_seq(seqs);
[pos5, score5] = svm_position(x5,out5, seqno5, endbulges, lenp);
% infer probabilities
[yside, yprec2] = interpolate_prob_new(score5, fit_filename);
%write to file
res = [seq_id; pos5; score5; yprec2; yside];
fid = fopen(outfile,'w');
fprintf(fid, '%d %d %g %g %g\n', res);
fclose(fid);
function svm_predict_();
%svm_predict_b(infile,outfile);
%perform svm position prediction
% version for large input files
% reads seqtot sequences at a time and classifies them
```

```
infile='c:\svm\in\draw_file.dat';
outfile='c:\svm\out\dicer_res.dat';
cd \\rosetta4\Development\gideon\svm\util
svm_light_folder = 'C:/svm/bin/';
model_filename5 = [svm_light_folder 'model5-2-6-2-6-21'];
tst_filename5 = 'C:/svm/Temp/svm_tst_5.dat';
svm_out_filename5 = 'C:/svm/Temp/out5.out';
fit_filename = 'C:/svm/Score/fit_p5-2-6-2-6-21.txt';
fidin = fopen(infile,'r');
fidout = fopen(outfile,'w');
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
   disp('reading structure...');
 [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);
 lenp = length_seq(seqs);

  disp('preprocessing and writing...');
 [x5, seqno5] = preprocess_and_write_data5(seqs,bulges1, ...
    bulges2,endbulges, tst_filename5);
  cd c:\
  dos([svm_light_folder 'svm_classify ' tst_filename5 ' ' model_filename5 ' ' svm_out_filename5]);
  cd \\rosetta4\Development\gideon\svm\util
% load and postprocess
 fidsvm = fopen(svm_out_filename5,'r');
 out5 = fscanf(fidsvm, '%g');
   fclose(fidsvm);

  disp('postprocessing...');
 [pos5, score5] = svm_position(x5,out5, seqno5, endbulges, lenp);
% infer probabilities
 [yside, yprec2] = interpolate_prob_new(score5, fit_filename);
%write to file
%seq_id0 is added so as to sequential order of sequence numbers
 seq_id = seq_id + seq_id0;
 res = [seq_id; pos5; score5; yprec2; yside];
   fprintf(fidout, '%d %d %g %g %g ', res);
   seq_id0 = max(seq_id);
end
fclose(fidin);
fclose(fidout);
quit
function svm_predict_b(infile,outfile);
%svm_predict_b(infile,outfile);
%perform svm position prediction
% version for large input files
% reads seqtot sequences at a time and classifies them
svm_light_folder = 'C:/svm/bin/';
model_filename5 = [svm_light_folder 'model5-2-6-2-6-21'];
tst_filename5 = 'C:/svm/Temp/svm_tst_5.dat';
```

```
svm_out_filename5 = 'C:/svm/Temp/out5.out';
fit_filename = 'C:/svm/Score/fit_p5-2-6-2-6-21.txt';
fidin = fopen(infile,'r');
fidout = fopen(outfile,'w');
seqstot = 1000; %number of sequences to classify each loop
seq_id0 = 0;
while ~feof(fidin)
   disp('reading structure...');
 [seqs,bulges1,bulges2,endbulges,seq_id] = read_structure_fid(fidin, seqstot);
 lenp = length_seq(seqs);

  disp('preprocessing and writing...');
 [x5, seqno5] = preprocess_and_write_data5(seqs,bulges1, ...
    bulges2,endbulges, tst_filename5);
 dos([svm_light_folder 'svm_classify ' tst_filename5 ' ' model_filename5 ' ' svm_out_filename5]);
% load and postprocess
 fidsvm = fopen(svm_out_filename5,'r');
 out5 = fscanf(fidsvm, '%g');
   fclose(fidsvm);

  disp('postprocessing...');
 [pos5, score5] = svm_position(x5,out5, seqno5, endbulges, lenp);
% infer probabilities
 [yside, yprec2] = interpolate_prob_new(score5, fit_filename);
%write to file
%seq_id0 is added so as to sequential order of sequence numbers
 seq_id = seq_id + seq_id0;
 res = [seq_id; pos5; score5; yprec2; yside];
   fprintf(fidout, '%d %d %g %g %g\n', res);
   seq_id0 = max(seq_id);
end
fclose(fidin);
fclose(fidout);
function unique_seqs(seqs,seqsd,bulges1, bulges2, endbulges, lenp, lend, pos, seq_id)
[y,l] = sort(lenp);
 bulges1 = bulges1(l);
 bulges2 = bulges2(l);
 endbulges = endbulges(l);
 lend = lend(l);
 lenp = lenp(l);
 pos = pos(l);
 seq_id = seq_id(l);
 seqs = seqs(l);
  seqsd = seqsd(l);

   count = 1;
   lc(count) = 1;
   for i = 2:length(seqs)
      if lenp(i) == lenp(i-1)
         if any(seqs{i} ~= seqs{i-1})
```

```
        count = count+1;
        lc(count) = i;
        end
      else
        count = count+1;
        lc(count) = i;
   end
   end

   keyboard
function write_examples(xi,yi, fid);
% %low level function
% write examples in format compatible with svm light
% xi,yi are the example vector + targets
% in testing mode the yi's are set to 0
for j = 1:size(xi,1)
    fprintf(fid,'%d',yi(j));
    I = find(xi(j,:));
    xprint = [I;xi(j,I)];
    fprintf(fid,' %d:%g',xprint);
    fprintf(fid,'\n');
endfunction write_examples_simple(xi,yi, fid);
% %low level function
% write examples in format compatible with svm light
% xi,yi are the example vector + targets
% in testing mode the yi's are set to 0
for j = 1:size(xi,1)
    fprintf(fid,'%d',yi(j));
    fprintf(fid,' %g',xi);
    fprintf(fid,'\n');
endkkk = 1:6000;
[x12_5, seqno_5] =
preprocess_and_write_data5(seqs(kk),bulges1(kk),bulges2(kk),endbulges(kk),'g:\research\rosetta\svm_light_utils1\sv
m_preprocessed\svm5_kk1.dat');
[x12_3, seqno_3] =
preprocess_and_write_data5(seqs(kk),bulges1(kk),bulges2(kk),endbulges(kk),'g:\research\rosetta\svm_light_utils1\sv
m_preprocessed\svm3_kk1.dat');
% use svm light to classify both each ewth its own model
load output5_kk1.out
load output3_kk1.out
[pos53kk, score53kk] = svm_position53(x12_5,output5_kk1, seqno_5, x12_3, output3_kk1, seqno_3, endbulges(kk),
lenp(kk));
%write final results to file
res = [seq_id(kk); pos53kk(:,1)'; pos53kk(:,2)'; score53kk'];
fid = fopen('res53_33156.out','a');
fprintf(fid, '%d %d %d %g\n', res);
fclose(fid);
```

```
function [first_pos,first_score] = firstk_determine(seqsd,seqs, pos1, pos2,ktup,k,range);
%[first_pos,first_score] = firstk_determine(seqsd,seqs, pos1, pos2,ktup,k,range);
%for each%for each palindrome with two mir predictions - which is better
%search on -2-+2 positions on each side for the best firstk score
if nargin ==4
   model.ktup = 8;
   model.k = 1;
   model.range = -2:2;
else
   model.ktup = ktup;
   model.k = k;
   model.range = range;
end
model.beta = 2;
model.use_min = 0;
first_pos = zeros(size(seqs));
first_score = zeros(size(seqs));
for i = 1:length(seqs)
   if mod(i,100) == 0, fprintf('..%d',i);end
   [first_pos(i), first_score(i)] = firstk_determine1(seqsd,seqs{i}, pos1(i), pos2(i), model);
end
fprintf('\n');
return
function [first_posi, first_scorei] = firstk_determine1(seqsd,seqsi, pos1i, pos2i,model);
k = model.k;
ktup = model.ktup;
beta = model.beta;
range = model.range;
%around pos1
for i = 1:length(range)
   posi = pos1i+range(i);
   if posi >0 & posi +ktup <= length(seqsi)
    p = seqsi(posi:posi+ktup-1);
    for j = 1:length(seqsd)
       d(j) = editD(p,seqsd{j}(1:ktup));
      end
    min_d1(i) = min(d);
   % take also the mean of highest percentile
    [ds,I] = sort(d);
      mean_d1(i) = mean(ds(1:k));
   else
      mean_d1(i) = nan;
   end
end
max1 = 1-min(mean_d1)/ktup;
%around pos2
for i = 1:length(range)
   posi = pos2i+range(i);
   if posi >0 & posi +ktup <= length(seqsi)
    p = seqsi(posi:posi+ktup-1);
```

```
    for j = 1:length(seqsd)
      d(j) = editD(p,seqsd{j}(1:ktup));
     end
    min_d2(i) = min(d);
   % take also the mean of highest percentile
    [ds,I] = sort(d);
     mean_d2(i) = mean(ds(1:k));
   else
     mean_d2(i) = nan;
   end
end
max2 = 1-min(mean_d2)/ktup;
if max1 > max2
   %first_posi = pos1i;
   first_posi = 1;
   first_scorei = max1;
elseif max2 > max1
   %first_posi = pos2i;
   first_posi = 2;
   first_scorei = max2;
else
   if model.use_min
     if min(min_d1) == min(min_d2)
      first_posi = nan;
      first_scorei = nan;
    elseif min(min_d1) < min(min_d2)
      first_posi = 1;
      first_scorei = 0;
   else
      first_posi = 2;
      first_scorei = 0;
     end
   else
     first_posi = nan;
     first_scorei = nan;
   end  %if model.use_min
end
return
load vars_hmdc440
seqs = palseq;
seqsd = mirseq;
pos = mirpos;
lend = mirlen;
clear palseq mirseq mirpos mirlen curdir datadir
function [id, palgrade5, pal_seq, pos1, pos2, score] = read_table_res(filename)
%[id, palgrade5, pal_seq, pos1, pos2, score] = read_table_res(filename)
fid = fopen(filename,'r');
k = 1;
while ~feof(fid);
   if (mod(k,100) == 0) fprintf('.'); end
```

```
  line = fgetl(fid);
  if line(1) ~= '%'
    [t,r] = strtok(line);
    id(k) = str2num(t);
    [t,r] = strtok(r);
    palgrade5(k) = str2num(t);
    [t,r] = strtok(r);
    pal_seq{k} = t;
    [t,r] = strtok(r);
    pos1(k) = str2num(t);
    [t,r] = strtok(r);
    pos2(k) = str2num(t);
    [t,r] = strtok(r);
    score(k) = str2num(t);
    k = k+1;
  end
end
fclose(fid);
return
function run_firstk_side_determine(filein, fileout)
%run_firstk_side_determine(filein, fileout)
%determine the betterr side of palgrade predictions, based of firstk
[id, palgrade5, pal_seq, pos1, pos2, score] = read_table_res(filein);
disp(['read  ' num2str(length(id)) '  records']);
load_all;
seqsd_all = transform_format(seqsd);
%remove records with nan positions
I = find(isnan(pos1) | isnan(pos2));
id(I) = [];
palgrade5(I) = [];
pal_seq(I) = [];
pos1(I) = [];
pos2(I) = [];
score(I) = [];
disp([num2str(length(id)) '  non null records passed to firstk_determine']);
fid = fopen(fileout,'w');
[first_pos,first_score] = firstk_determine(seqsd_all,pal_seq, pos1, pos2,10, 3, -1:1);
res = [id; first_pos; first_score];
fprintf(fid, '%d %d %5.3f\n',res);
fclose(fid);
return
```

```
function bs_scoring_all_zukers_v5(infilename,outfilename)
% function bs_scoring_all_zukers_v5(infilename,outfilename)
% for each binding site, gives the score of all of its zuker versions, each
% score is between 0 and 1, where 1 is highest score.
% writes to the file outfilename in the format: bs_id zuker_version score bs_id...
% bs_id is as it appears in the input file (named infilename)
% see the paramfile and README for explanations on parameters
paramfilename = 'params5';
bs_tot = 1000;
eval(paramfilename);
fidout = fopen(outfilename,'w');
fidin = fopen(infilename,'r');
fprintf(fidout,'%%each line contains the following info:\r\n');
fprintf(fidout,'%%bs_id zuker_version score ratio_paired num_mir_bulges num_target_bulges sum_mir_tail_lens
bulge_kernel_mir num_gts\r\n');
while ~feof(fidin)
  a = read_brzuker(fidin,bs_tot);
  for i = 1:length(a)
    this_bs = a(i);
    for j=1:this_bs.numzukers
      z = this_bs.zukers(j);
      [s,f,ratio_paired,bulge_K,num_gts] = get_zuker_score_and_features(z,this_bs,params);
      if(f.has_draw == 0) % no draw available
        res = [nan nan nan nan nan nan];
      else
        res = [ratio_paired, f.num_mir_bulges, f.num_target_bulges,...
            (f.mir_tail5_len+f.mir_tail3_len), bulge_K, num_gts];
      end
      fprintf(fidout,'%d %d %g %g %d %d %d %g %d\r\n',this_bs.bs_id,z.version,s,res);
    end
  end
end
fclose(fidin);
fclose(fidout);
function f_struct = features_from_zuker(zuker)
% function f_struct = features_from_zuker(zuker)
% zuker(i).draw_line1 is the string of the first of the 4 lines in the draw
% same for line2 thru line4. If no draw exists (rnastructure failed to draw)
% all lines are an 'X', and f.has_draw = 0. If there is a draw
% f.has_draw = 1, and f includes all the features extracted from the draw.
% f_struct(i) is a collection of fields describing this zuker.
% if there is only one zuker, f_struct is a single struct.
single_flag = 0;
if(length(zuker)==1)
  single_flag = 1;
  tt(1) = zuker;
  zuker = tt;
end
for i=1:length(zuker)
  z = zuker(i);
```

```
% --------------------------------
% correct lengths of strings to all same length
line1 = z.draw_line1;
line2 = z.draw_line2;
line3 = z.draw_line3;
line4 = z.draw_line4;
if(strcmp(upper(line1(1)),'X'))
  f.has_draw = 0;
else
  f.has_draw = 1;
  f.energy = z.energy;
  tt = max([find(isletter(line1)),find(isletter(line2)),...
       find(isletter(line3)),find(isletter(line4))]);
  line1 = [line1(1:min(tt,length(line1)))];
  for k=1:tt-length(line1); line1=[line1,' ']; end
  line2 = [line2(1:min(tt,length(line2)))];
  for k=1:tt-length(line2); line2=[line2,' ']; end
  line3 = [line3(1:min(tt,length(line3)))];
  for k=1:tt-length(line3); line3=[line3,' ']; end
  line4 = [line4(1:min(tt,length(line4)))];
  for k=1:tt-length(line4); line4=[line4,' ']; end
  % --------------------------------
  % find bulges and windows by order. bulge,win,bulge win etc
  paired_wins_lens = [];
  mir_bulge_vec = []; % of length as mir. 1 if on bulge, 0 else.
  target_bulge_vec = []; % same for target.
  if(isletter(line2(1)))
    in_bulge = 0;
    mir_bulges_lens = 0;
    target_bulges_lens = 0;
    this_win_len = 1;
  else
    in_bulge = 1;
    if(isletter(line1(1)))
      this_mir_bulge = 1;
    else
      this_mir_bulge = 0;
    end
    if(isletter(line4(1)))
      this_target_bulge = 1;
    else
      this_target_bulge = 0;
    end
    mir_bulges_lens = [];
    target_bulges_lens = [];
  end
  for k=2:length(line1)
    if(isletter(line2(k)))
      if(in_bulge)
        in_bulge = 0;
```

```
        mir_bulges_lens = [mir_bulges_lens,this_mir_bulge];
        target_bulges_lens = [target_bulges_lens,this_target_bulge];
        this_win_len = 1;
      else
        this_win_len = this_win_len + 1;
      end
    else
      if(in_bulge)
        if(isletter(line1(k)))
          this_mir_bulge = this_mir_bulge + 1;
        end
        if(isletter(line4(k)))
          this_target_bulge = this_target_bulge + 1;
        end
      else
        in_bulge = 1;
        paired_wins_lens = [paired_wins_lens,this_win_len];
        if(isletter(line1(k)))
          this_mir_bulge = 1;
        else
          this_mir_bulge = 0;
        end
        if(isletter(line4(k)))
          this_target_bulge = 1;
        else
          this_target_bulge = 0;
        end
      end
    end
  end
  if(isletter(line2(end))) % finished in paired win
    mir_bulges_lens = [mir_bulges_lens,0];
    target_bulges_lens = [target_bulges_lens,0];
    paired_wins_lens = [paired_wins_lens,this_win_len];
  else
    mir_bulges_lens = [mir_bulges_lens,this_mir_bulge];
    target_bulges_lens = [target_bulges_lens,this_target_bulge];
  end
  % get mir and target bulge vecs
  mir_bulge_nonsym_vec = [];
  mir_bulge_sym_vec = [];
  target_bulge_nonsym_vec = [];
  target_bulge_sym_vec = [];
  L = length(mir_bulges_lens);
  for k = 1:L
    ml = mir_bulges_lens(k);
    tl = target_bulges_lens(k);
    if(ml>0 & tl>0) % symmetric bulge
      mir_bulge_nonsym_vec = [mir_bulge_nonsym_vec,zeros(1,ml)];
      mir_bulge_sym_vec = [mir_bulge_sym_vec,ones(1,ml)];
```

```
        target_bulge_nonsym_vec = [target_bulge_nonsym_vec,zeros(1,tl)];
        target_bulge_sym_vec = [target_bulge_sym_vec,ones(1,tl)];
      elseif(ml>0 & tl==0)
        mir_bulge_nonsym_vec = [mir_bulge_nonsym_vec,ones(1,ml)];
        mir_bulge_sym_vec = [mir_bulge_sym_vec,zeros(1,ml)];
      elseif(ml==0 & tl>0)
        target_bulge_nonsym_vec = [target_bulge_nonsym_vec,ones(1,tl)];
        target_bulge_sym_vec = [target_bulge_sym_vec,zeros(1,tl)];
      else
      end
      if(k<L) % there is window after
        wl = paired_wins_lens(k);
        mir_bulge_nonsym_vec = [mir_bulge_nonsym_vec,zeros(1,wl)];
        mir_bulge_sym_vec = [mir_bulge_sym_vec,zeros(1,wl)];
        target_bulge_nonsym_vec = [target_bulge_nonsym_vec,zeros(1,wl)];
        target_bulge_sym_vec = [target_bulge_sym_vec,zeros(1,wl)];
      end
    end
    % --------------------------------
    % update related features in f.
    f.paired_wins_lens_direction_5to3_onmir = paired_wins_lens;
    f.num_paired_wins = sum(paired_wins_lens>0);
    f.mir_bulges_lens_5to3 = mir_bulges_lens;
    f.num_mir_bulges = sum(mir_bulges_lens>0);
    f.target_bulges_lens_3to5 = target_bulges_lens;
    f.num_target_bulges = sum(target_bulges_lens>0);
    f.unified_bulges_lens = mir_bulges_lens + target_bulges_lens;
    f.mir_tail5_len = mir_bulges_lens(1);
    f.mir_tail3_len = mir_bulges_lens(end);
    f.target_tail5_len = target_bulges_lens(end);
    f.target_tail3_len = target_bulges_lens(1);
    f.total_nucs = sum(2*paired_wins_lens) + ...
      sum(mir_bulges_lens) + sum(target_bulges_lens);
    f.num_nucs_paired = sum(2*paired_wins_lens);
    f.mir_nonsym_bulge_vec_5to3 = mir_bulge_nonsym_vec;
    f.mir_sym_bulge_vec_5to3 = mir_bulge_sym_vec;
    f.mir_bulge_vec_5to3 = mir_bulge_nonsym_vec + mir_bulge_sym_vec;
    f.target_nonsym_bulge_vec_3to5 = target_bulge_nonsym_vec;
    f.target_sym_bulge_vec_3to5 = target_bulge_sym_vec;
    f.target_bulge_vec_5to3 = target_bulge_nonsym_vec + target_bulge_sym_vec;
    % --------------------------------
  end % if strcmp
  f_struct(i) = f;
end % end loop on length(zuker)
if(single_flag)
  tt = f_struct(1);
  f_struct = tt;
end
function [score,f,ratio_paired,bulge_K,num_gts] = get_zuker_score_and_features(z,bs,params)
f = features_from_zuker(z); % f is a struct holding many features
```

```
if(f.has_draw == 0) % no draw available - give a score of 0!
  score = nan;
  ratio_paired = nan;
  bulge_K = nan;
  num_gts = nan;
  return;
end
ratio_paired = f.num_nucs_paired/f.total_nucs;
% normalize weights to sum of 1:
sum_ws = params.w_energy + params.w_ratio_paired + params.w_num_mir_bulges + ...
  params.w_num_target_bulges + params.w_mir_tail_lens + params.w_target_tail_lens + ...
  params.w_mir_bulge_kernel + params.w_gt_pairs;
w_energy = params.w_energy/sum_ws;
w_ratio_paired = params.w_ratio_paired/sum_ws;
w_num_mir_bulges = params.w_num_mir_bulges/sum_ws;
w_num_target_bulges = params.w_num_target_bulges/sum_ws;
w_mir_tail_lens = params.w_mir_tail_lens/sum_ws;
w_target_tail_lens = params.w_target_tail_lens/sum_ws;
w_mir_bulge_kernel = params.w_mir_bulge_kernel/sum_ws;
w_gt_pairs = params.w_gt_pairs/sum_ws;
[score_bk,bulge_K] = score_bulge_kernel(f.mir_bulge_vec_5to3,params.mir_bulge_pos_prices);
[score_gt,num_gts] = score_gt_pairs(z,params);
score = w_energy * min_max_score(params.min_energy,params.max_energy,-1,z.energy) + ...
  w_ratio_paired * ratio_paired + ...
  w_num_mir_bulges * min_max_score(params.min_num_mir_bulges,...
  params.max_num_mir_bulges,-1,f.num_mir_bulges) + ...
    w_num_target_bulges * min_max_score(params.min_num_target_bulges,...
  params.max_num_target_bulges,-1,f.num_target_bulges) + ...
    w_mir_tail_lens * min_max_score(params.min_mir_tail_lens,...
  params.max_mir_tail_lens,-1,(f.mir_tail5_len+f.mir_tail3_len)) + ...
    w_target_tail_lens * min_max_score(params.min_target_tail_lens,...
  params.max_target_tail_lens,-1,(f.target_tail5_len+f.target_tail3_len)) + ...
    w_mir_bulge_kernel * score_bk + w_gt_pairs * score_gt;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
function score = min_max_score(min_v,max_v,dir_flag,value)
if(dir_flag == 1) % the higher the better
  score = (value - min_v)/(max_v - min_v);
elseif(dir_flag == -1) % the lower the better
  score = 1 - ((value - min_v)/(max_v - min_v));
else
  error('min_max_score: dir_flag must be 1 or -1. aborting');
end
if(score<0)
  score = 0;
  warning('min_max_score: encountered value outside range getting neg score. truncating score to 0');
end
if(score>1)
  score = 1;
  warning('min_max_score: encountered value outside range getting score higher than 1. truncating score to 1');
```

```
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
function [score,bulge_K] = score_bulge_kernel(bulge_vec,K)
l_vec = length(bulge_vec);
l_K = length(K);
r = floor(l_vec/l_K);
kernel = [];
for i = 1:l_K
  kernel = [kernel,K(i)*ones(1,r)];
end
kernel = [kernel,K(end)*ones(1,l_vec-r*l_K)];
kernel = kernel/sum(kernel);
bulge_K = sum(bulge_vec .* kernel); % weighted sum of bulges by position on vec
score = min_max_score(0,1,-1,bulge_K);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
function [score,num_gts] = score_gt_pairs(z,params)
num_gts = 0;
for i=1:length(z.draw_line2)
  t1 = z.draw_line2(i);
  t2 = z.draw_line3(i);
  if((strcmp('G',t1) & strcmp('T',t2)) | (strcmp('T',t1) & strcmp('G',t2)))
    num_gts = num_gts + 1;
  end
end
score = min_max_score(params.min_num_gts,params.max_num_gts,-1,num_gts);params.choose_zuker_by = 'max';
% relevant only for bs_scoring_single_zuker
% range of energies. If the minimum is -30 and the max is 0 then the energy score of a
% zuker having energy -10 is 1/3. -30 will get a score of 1 etc. The score will be
% weighed also by w_energy.
params.max_energy = 0;
params.min_energy = -30;
% as above but for number of bulges
params.min_num_mir_bulges = 0;
params.max_num_mir_bulges = 6;
params.min_num_target_bulges = 0;
params.max_num_target_bulges = 6;
% again....
params.min_mir_tail_lens = 0;
params.max_mir_tail_lens = 15;
params.min_target_tail_lens = 0;
params.max_target_tail_lens = 15; % by default not using it anyways
% again...
params.min_num_gts = 0;
params.max_num_gts = 6;
% gives the pricing on bulges depending on position on mir
% if the vector is [1,0,1] then bulged nucs in the first and third third of the mir
% are penalized, while those in the middle third are not at all (basically
% a weighted sum).
```

```
% do not worry about normalization of this:
params.mir_bulge_pos_prices = [1,0,1];
% below the weights of each feature in the total score. The score of
% a zuker version is the sum of each of the individual scores times its
% weight. bs_scoring normalizes the sum of these to 1 (so here take care only of ratios)
params.w_energy = 1;
params.w_ratio_paired = 4;
params.w_num_mir_bulges = 0.5;
params.w_num_target_bulges = 0.5;
params.w_mir_tail_lens = 2;
params.w_target_tail_lens = 0; % should be 0 unless you have a very good reason to change
params.w_gt_pairs = 1;
params.w_mir_bulge_kernel = 1;
function brzuker_out_data = read_brzuker(fid,bs_tot)
% function brzuker_out_data = read_brzuker(fid,bs_tot)
% reads bs_tot binding sites at a time
% format of file:
% for each candidate binding site:
% > ofir's internal bs_id
% mirseq
% target seq
% 1 (which zuker version)
% energy
% 4 lines describing the zuker draw
% brzuker_out_data(x) is a struct describing the data for the xth bs
% its feilds are:
% mir id ; utr id; offset; bs_id; mirseq; targetseq; mirlen; targetlen;
% numzukers; zukers;
% zukers{j} is again a struct with the feilds:
% energy; 4 lines of zuker draw
% numzukers is how many different zuker folds it found
counter = 0;
while (~feof(fid) & (counter < bs_tot))
  bp1 = [];
  bp2 = [];
  counter = counter + 1;
  tt = fscanf(fid,'>%d\n');
  od(counter).bs_id = tt(1);
  od(counter).mirseq = fgetl(fid);
  this_mirlen = length(od(counter).mirseq);
  od(counter).mirlen = this_mirlen;
  od(counter).targetseq = fgetl(fid);
  this_targetlen = length(od(counter).targetseq);
  od(counter).targetlen = this_targetlen;
  nz = 0;
  next_line = fgetl(fid);
  new_bs = 0;
  while (new_bs==0)
    nz = nz+1;
    this_z.version = str2double(next_line);
```

```
      e = str2double(fgetl(fid));
      this_z.energy = e;
      line1 = fgetl(fid);
      line2 = fgetl(fid);
      line3 = fgetl(fid);
      line4 = fgetl(fid);
      this_z.draw_line1 = line1;
      this_z.draw_line2 = line2;
      this_z.draw_line3 = line3;
      this_z.draw_line4 = line4;
      od(counter).zukers(nz) = this_z;
      next_line = fgetl(fid);
      if(strcmp(next_line,'|'))
        new_bs = 1;
      end
    end
    od(counter).numzukers = nz;
end
brzuker_out_data = od;
function brzuker_out_data = raed_brzuker(fid,bs_tot)
% function brzuker_out_data = raed_brzuker(fid,bs_tot)
% reads output from Baraks brzuker program. raeds bs_tot binding sites at a time
% format of file:
% for each candidate binding site:
% > ofir's internal bs_id
% mirseq
% target seq
% 1 (which zuker version)
% energy
% 4 lines describing the zuker draw
% brzuker_out_data(x) is a struct describing the data for the xth bs
% its feilds are:
% mir id ; utr id; offset; bs_id; mirseq; targetseq; mirlen; targetlen;
% numzukers; zukers;
% zukers{j} is again a struct with the feilds:
% energy; 4 lines of zuker draw
% numzukers is how many different zuker folds it found
counter = 0;
while (~feof(fid) & (counter < bs_tot))
    bp1 = [];
    bp2 = [];
    counter = counter + 1;
    tt = fscanf(fid,'>%d\n');
    od(counter).bs_id = tt(1);
    od(counter).mirseq = fgetl(fid);
    this_mirlen = length(od(counter).mirseq);
    od(counter).mirlen = this_mirlen;
    od(counter).targetseq = fgetl(fid);
    this_targetlen = length(od(counter).targetseq);
    od(counter).targetlen = this_targetlen;
```

```
  nz = 0;
  next_line = fgetl(fid);
  new_bs = 0;
  while (new_bs==0)
    nz = nz+1;
    this_z.version = str2num(next_line);
    e = str2num(fgetl(fid));
    this_z.energy = e;
    line1 = fgetl(fid);
    line2 = fgetl(fid);
    line3 = fgetl(fid);
    line4 = fgetl(fid);
    this_z.draw_line1 = line1;
    this_z.draw_line2 = line2;
    this_z.draw_line3 = line3;
    this_z.draw_line4 = line4;
    od(counter).zukers(nz) = this_z;
    next_line = fgetl(fid);
    if(strcmp(next_line,'|'))
      new_bs = 1;
    end
  end
  od(counter).numzukers = nz;
end
brzuker_out_data = od;
```

```
function [xs,ys,xp2,yp2] = analyse_errors_bins2(pos_estimated,score,pos, endbulges,N)
% measure the distribution of erros
if length(pos_estimated) ~= length(score)
   error('pos_estimated  and  score not compatible');
end
if length(pos_estimated) ~= length(pos)
   error('pos_estimated  and  pos not compatible');
end
if length(pos_estimated) ~= length(endbulges)
   error('pos_estimated  and  endbulges size not compatible');
end
if nargin == 4
   N = 6;
end
perc = [1:-1/N:0]*100;
thresh = prctile(score, perc);
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
   eb = find(endbulges{i});
   correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)-1
   I = find(score <= thresh(i) & score >= thresh(i+1));
   if ~isempty(I)
      count = count + 1;
      midbin(count) = mean(score(I));
      accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

      J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
      correct_side_dist1(count) = length(J1)/length(I);
      J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
      correct_side_dist2(count) = length(J2)/length(I);
      J3 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 3);
      correct_side_dist3(count) = length(J3)/length(I);
      Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 3);
      correct_side_disth(count) = length(Jh)/length(I);

      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
   else
      count = count+1;
      midbin(count) = NaN;;
```

```
    accuracy(count) = NaN;
    correct_side_dist1(count) = NaN;
    correct_side_dist2(count) = NaN;
    correct_side_disth(count) = NaN;
    wrong_side(count) = NaN;
    fraction(count) = NaN;
  end
end
acc1 = accuracy + correct_side_dist1;
acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
acc3 = accuracy + correct_side_dist1 + correct_side_dist2 + correct_side_dist3;
hold on
plot(midbin, acc3,'y','linewidth',2)
plot(midbin, acc2,'g','linewidth',2)
plot(midbin, acc1,'r','linewidth',2)
plot(midbin, accuracy,'b','linewidth',2)
plot(midbin, wrong_side,'k','linewidth',2)
plot(midbin,fraction,'c','linewidth',2)
legend('dist \leq 3', 'dist \leq 2', 'dist \leq 1', 'precise', 'wrong side',2);
plot(midbin, acc3,'dy')
plot(midbin, acc2,'*g')
plot(midbin, acc1,'or')
plot(midbin, accuracy,'bd')
plot(midbin, wrong_side,'kv')
xlabel('bin');
%axis([min(midbin)-1 max(midbin)+1 0 1])
[ry,yp2,mass,xp2,newy,pos] = isotonic_regression(midbin,acc2);
[ry,ys,mass,xs,newy,pos] = isotonic_regression(midbin,1-wrong_side);
returnfunction [x,ys,yp2,yp1,yp0] = analyse_errors_bins3(pos_estimated,score,pos, endbulges,N)
% measure the distribution of erros
if length(pos_estimated) ~= length(score)
  error('pos_estimated  and  score not compatible');
end
if length(pos_estimated) ~= length(pos)
  error('pos_estimated  and  pos not compatible');
end
if length(pos_estimated) ~= length(endbulges)
  error('pos_estimated  and  endbulges size not compatible');
end
if nargin == 4
  N = 6;
end
perc = [1:-1/N:0]*100;
thresh = prctile(score, perc);
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
```

```
count = 0;
N = length(pos);
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)-1
  I = find(score <= thresh(i) & score >= thresh(i+1));
  if ~isempty(I)
    count = count + 1;
    midbin(count) = mean(score(I));
    accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

    J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
    correct_side_dist1(count) = length(J1)/length(I);
    J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
    correct_side_dist2(count) = length(J2)/length(I);
    J3 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 3);
    correct_side_dist3(count) = length(J3)/length(I);
    Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 3);
    correct_side_disth(count) = length(Jh)/length(I);

    wrong_side(count) = sum(1-correct_side(I))/length(I);

    fraction(count) = length(I)/N;
  else
    count = count+1;
    midbin(count) = NaN;;
    accuracy(count) = NaN;
    correct_side_dist1(count) = NaN;
    correct_side_dist2(count) = NaN;
    correct_side_disth(count) = NaN;
    wrong_side(count) = NaN;
    fraction(count) = NaN;
  end
end
acc1 = accuracy + correct_side_dist1;
acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
acc3 = accuracy + correct_side_dist1 + correct_side_dist2 + correct_side_dist3;
hold on
plot(midbin, acc3,'y','linewidth',2)
plot(midbin, acc2,'g','linewidth',2)
plot(midbin, acc1,'r','linewidth',2)
plot(midbin, accuracy,'b','linewidth',2)
plot(midbin, wrong_side,'k','linewidth',2)
plot(midbin,fraction,'c','linewidth',2)
legend('dist \leq 3', 'dist \leq 2', 'dist \leq 1', 'precise', 'wrong side',2);
plot(midbin, acc3,'dy')
plot(midbin, acc2,'*g')
plot(midbin, acc1,'or')
```

```
plot(midbin, accuracy,'bd')
plot(midbin, wrong_side,'kv')
xlabel('bin');
%axis([min(midbin)-1 max(midbin)+1 0 1])
[ry,yp2,mass,xp2,newy,pos] = isotonic_regression(midbin,acc2);
[ry,yp1,mass,xp1,newy,pos] = isotonic_regression(midbin,acc1);
[ry,yp0,mass,xp0,newy,pos] = isotonic_regression(midbin,accuracy);
[ry,ys,mass,xs,newy,pos] = isotonic_regression(midbin,1-wrong_side);
x=xs;
returnfunction res = analyse_errors_perc(pos_estimated,score,pos, endbulges)
%analyse_errors_perc(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
N = 100;
perc = [1:-1/N:0]*100;
thresh = prctile(score, perc);
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)
   I = find(score >= thresh(i));
   if ~isempty(I)
      count = count + 1;
      accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

      J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
      correct_side_dist1(count) = length(J1)/length(I);
      J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
      correct_side_dist2(count) = length(J2)/length(I);
      J3 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 3);
      correct_side_dist3(count) = length(J3)/length(I);
      Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 3);
      correct_side_disth(count) = length(Jh)/length(I);

      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
   else
      count = count+1;
      accuracy(count) = NaN;
      correct_side_dist1(count) = NaN;
      correct_side_dist2(count) = NaN;
```

```
      correct_side_disth(count) = NaN;
      wrong_side(count) = NaN;
      fraction(count) = NaN;
    end
  end

  acc1 = accuracy + correct_side_dist1;
  acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
  acc3 = accuracy + correct_side_dist1 + correct_side_dist2 + correct_side_dist3;
  %clf
  hold on

  plot(perc, acc3,'y','linewidth',2)
  plot(perc, acc2,'g','linewidth',2)
  plot(perc, acc1,'r','linewidth',2)
  plot(perc, accuracy,'b','linewidth',2)
  plot(perc, wrong_side,'k','linewidth',2)
  plot(perc, thresh,'c','linewidth',2)
  legend('dist \leq 3','dist \leq 2', 'dist \leq 1', 'precise', 'wrong side', 'threshold',2);
  xlabel('percentage');
  axis([0 100 0 1]);

  %keyboard
  %prepare result
  N = length(accuracy);
  res = [accuracy(N), acc1(N), acc2(N), acc3(N), 1-wrong_side(N), acc2(round(0.2*N))]
  returnfunction analyse_errors_perc_2preds(pos_estimated,score,pos, endbulges,decide_by,pred_side)
%analyse_errors_perc(pos_estimated,score,pos, endbulges)
%pos_est and score are 2 cols (a pred for each arm with its score)
if(~exist('decide_by'))
  decide_by = 0;
end
if(decide_by == 1) % decide on prediction using real mirside
  for i=1:length(pos)
    lb = find(endbulges{i});
    eb_begin = lb(1);
    eb_end = lb(end);
    mirside = (pos(i)<eb_begin); % mirside=1 if mir is on arm5, 0 if on arm3.
    if(mirside) % arm5
      pos_est_arm(i) = pos_estimated(i,1);
      score_arm(i) = score(i,1);
    else
      pos_est_arm(i) = pos_estimated(i,2);
      score_arm(i) = score(i,2);
    end
  end
elseif(decide_by == 0) % decide by best score
  for i=1:length(pos)
    if(score(i,1)>score(i,2))
      pos_est_arm(i) = pos_estimated(i,1);
```

```
      score_arm(i) = score(i,1);
    else
      pos_est_arm(i) = pos_estimated(i,2);
      score_arm(i) = score(i,2);
    end
  end
elseif(decide_by == 2) % decide by predicted side
  if(~exist('pred_side'))
    error('must give a predicted side for this option')
  end
  for i=1:length(pos)
    if(pred_side(i)==1) % arm5 predicted
      pos_est_arm(i) = pos_estimated(i,1);
      score_arm(i) = score(i,1);
    else
      pos_est_arm(i) = pos_estimated(i,2);
      score_arm(i) = score(i,2);
    end
  end
end
analyse_errors_perc(pos_est_arm,score_arm,pos,endbulges);
function res = analyse_errors_perc_noplot(pos_estimated,score,pos, endbulges)
%analyse_errors_perc(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
N = 100;
perc = [1:-1/N:0]*100;
thresh = prctile(score, perc);
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)
   I = find(score >= thresh(i));
   if ~isempty(I)
     count = count + 1;
     accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

     J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
     correct_side_dist1(count) = length(J1)/length(I);
     J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
     correct_side_dist2(count) = length(J2)/length(I);
     Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
```

```
      correct_side_disth(count) = length(Jh)/length(I);


      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
    else
      count = count+1;
      accuracy(count) = NaN;
      correct_side_dist1(count) = NaN;
      correct_side_dist2(count) = NaN;
      correct_side_disth(count) = NaN;
      wrong_side(count) = NaN;
      fraction(count) = NaN;
    end
  end

  acc1 = accuracy + correct_side_dist1;
  acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
  N = length(accuracy);
  res = [accuracy(N), acc1(N), acc2(N), 1-wrong_side(N), acc2(round(0.2*N))];
  returnfunction analyse_errors_thresh(pos_estimated,score,pos, endbulges,Np)
%analyse_errors_thresh(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
if max(score) > 1
   mxscore = max(score);
else
   mxscore = 1;
end
if min(score) < 0
   mnscore = min(score);
else
   mnscore = 0;
end
if(~exist('Np'))
   Np = 500;
end
dth = (mxscore- mnscore)/Np;
thresh = mnscore:dth:mxscore;
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
   eb = find(endbulges{i});
   correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
```

```
end
for i = 1:length(thresh)
   I = find(score >= thresh(i));
   if ~isempty(I)
      count = count + 1;
      accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

      J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
      correct_side_dist1(count) = length(J1)/length(I);
      J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
      correct_side_dist2(count) = length(J2)/length(I);
      Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
      correct_side_disth(count) = length(Jh)/length(I);


      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
   else
      count = count+1;
      accuracy(count) = NaN;
      correct_side_dist1(count) = NaN;
      correct_side_dist2(count) = NaN;
      correct_side_disth(count) = NaN;
      wrong_side(count) = NaN;
      fraction(count) = NaN;
   end
end

acc1 = accuracy + correct_side_dist1;
acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
clf
hold on

plot(thresh, acc2,'g')
plot(thresh, acc1,'r')
plot(thresh, accuracy,'b')
plot(thresh, wrong_side,'k')
plot(thresh, fraction,'c')
legend('dist \leq 2', 'dist \leq 1', 'precise', 'wrong side', 'fraction');
xlabel('threshold');
%keyboard
returnfunction [thresh,acc2,captures] = analyse_errors_thresh_B(pos_estimated,score,pos, endbulges,thresh)
%analyse_errors_thresh_B(pos_estimated,score,pos, endbulges,thresh)
% receives the vector thresh
% measure the distribution of erros
if max(score) > 1
   mxscore = max(score);
else
   mxscore = 1;
```

```
end
if min(score) < 0
  mnscore = min(score);
else
  mnscore = 0;
end
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)
  I = find(score >= thresh(i));
  captures(i) = length(I);
   if ~isempty(I)
     count = count + 1;
     accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

     J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
     correct_side_dist1(count) = length(J1)/length(I);
     J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
     correct_side_dist2(count) = length(J2)/length(I);
     Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
     correct_side_disth(count) = length(Jh)/length(I);


     wrong_side(count) = sum(1-correct_side(I))/length(I);

     fraction(count) = length(I)/N;
   else
     count = count+1;
     accuracy(count) = NaN;
     correct_side_dist1(count) = NaN;
     correct_side_dist2(count) = NaN;
     correct_side_disth(count) = NaN;
     wrong_side(count) = NaN;
     fraction(count) = NaN;
   end
end

acc1 = accuracy + correct_side_dist1;
acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
```

```
 hold on
 plot(thresh, acc2,'g-o','linewidth',2)
 plot(thresh, acc1,'r-o','linewidth',2)
 plot(thresh, accuracy,'b-o','linewidth',2)
 plot(thresh, wrong_side,'k-o','linewidth',2)
 plot(thresh, fraction,'c-o','linewidth',2)
 legend('dist \leq 2', 'dist \leq 1', 'precise', 'wrong side', 'fraction');
 xlabel('threshold');
 %keyboard
 returnfunction [thresh,captures,acc2,acc1,accuracy,correctside] = ...
   analyse_errors_thresh_C(pos_estimated,score,pos, endbulges,thresh)
% [thresh,captures,acc2,acc1,accuracy,correctside] = analyse_errors_thresh_C(pos_estimated,score,pos,
endbulges,thresh)
% receives the vector thresh
% measure the distribution of erros
if max(score) > 1
   mxscore = max(score);
else
   mxscore = 1;
end
if min(score) < 0
   mnscore = min(score);
else
   mnscore = 0;
end
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
   eb = find(endbulges{i});
   correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)
   I = find(score >= thresh(i));
   captures(i) = length(I);
    if ~isempty(I)
       count = count + 1;
       accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

       J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
       correct_side_dist1(count) = length(J1)/length(I);
       J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
       correct_side_dist2(count) = length(J2)/length(I);
       Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 2);
       correct_side_disth(count) = length(Jh)/length(I);
```

```
      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
    else
      count = count+1;
      accuracy(count) = NaN;
      correct_side_dist1(count) = NaN;
      correct_side_dist2(count) = NaN;
      correct_side_disth(count) = NaN;
      wrong_side(count) = NaN;
      fraction(count) = NaN;
    end
  end

  acc1 = accuracy + correct_side_dist1;
  acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
  correctside = 1-wrong_side;

  hold on
  plot(thresh, acc2,'g-o','linewidth',2)
  plot(thresh, acc1,'r-o','linewidth',2)
  plot(thresh, accuracy,'b-o','linewidth',2)
  plot(thresh, wrong_side,'k-o','linewidth',2)
  plot(thresh, fraction,'c-o','linewidth',2)
  legend('dist \leq 2', 'dist \leq 1', 'precise', 'wrong side', 'fraction');
  xlabel('threshold');
  %keyboard
  returnfunction mfe = anti_inds_to_mfe(anti_inds)
% anti_inds holds for each nuc in the seq what is the index of
% the nuc across from it where the 0 means unpaired (this is returned by read_structure_withanti).
% returns mfe which is the structure in the format of rnafold, i.e. only base pairs:
% mfe is a 2 col matrix, the first being the bases on arm5 which are paired and the second
% their corresponding pairs
if(~iscell(anti_inds))
  mfe = get_mfe(anti_inds);
  return;
end
for i=1:length(anti_inds)
  mfe{i} = get_mfe(anti_inds{i});
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%
function mfe = get_mfe(ai)
bps=0;
for i=1:length(ai)
  if(ai(i))
    if(i>ai(i))
      return
```

```
      end
      bps = bps+1;
      mfe(bps,1) = i;
      mfe(bps,2) = ai(i);
    end
end
function [sum_in_win, sum_in_win_mfe, sum_out, sum_out_mfe, faulty] = ...
    base_pairing(pal_len, bp_prob, mfe, winstart5, win_len)
% function [sum_in_win, sum_in_win_mfe, sum_out, sum_out_mfe, faulty] = ...
%     base_pairing(pal_len, bp_prob, mfe, winstart5, win_len)
% pal_len is length of palindrom
% bp_prob is the base pairing prob matrix which has 3 cols:
% 5side index, 3side index, prob to be paired
% mfe has the pairs in the min free energy drawing
% winstart5 is the positon of the start of the window in question
% win_len is its length
% sum_in_win is the sum of the bp probs of all pairs involving a base
% in the designated window normalized by win_len
% sum_in_win_mfe is the sum of the bp probs of all pairs appearing
% in the mfe structure and involving a base in the window. this is
% normalized by the number of base pairs appearing in the mfe structure
% within the window (if only one folding possible sum_in_win_mfe=1).
% sum_out is like sum_in only all bases not in window. normalized by
% ((pal_len-eb_len)/2 - win_len).
% sum_out_mfe is like sum_in_win_mfe only for all bp not in window.
% analogous normalization.
% if window is illegal, returns faulty=1 and NAN for other values
% also note that no check is made on winstart5 and win_len being positive (which they must) - beware!
n_pairs = size(bp_prob,1);
n_mfe_pairs = size(mfe,1);
arm5 = mfe(:,1);
arm3 = mfe(:,2);
eb_start = arm5(end)+1;
eb_end = arm3(end)-1;
eb_len = eb_end-eb_start+1;  % num nucs in end bulge
win_inds = [winstart5:winstart5+win_len-1];
if(any(intersect(win_inds,[eb_start:eb_end])) | win_inds(end)>pal_len)
  faulty = 1;
  sum_in_win = NaN;
  sum_in_win_mfe = NaN;
  sum_out = NaN;
  sum_out_mfe = NaN;
  disp('WINDOW IS ILLEGAL. RETURNING FAULTY=1.');
  disp(['window has ' num2str(length(intersect(win_inds,[eb_start:eb_end]))) ' nucs in endloop']);
  return
end
sum_in_win = 0;
sum_in_win_mfe = 0;
sum_out = 0;
sum_out_mfe = 0;
```

```
faulty = 0;
n_mfe_pairs_inwin = 0;
for i=1:n_pairs
  side5 = bp_prob(i,1);
  side3 = bp_prob(i,2);
  if(ismembc(side5,win_inds) | ismembc(side3,win_inds))
    sum_in_win = sum_in_win + bp_prob(i,3);
    if(ismember([side5,side3],mfe,'rows'))
      sum_in_win_mfe = sum_in_win_mfe + bp_prob(i,3);
      n_mfe_pairs_inwin = n_mfe_pairs_inwin + 1;
    end
  else
    sum_out = sum_out + bp_prob(i,3);
    if(ismember([side5,side3],mfe,'rows'))
      sum_out_mfe = sum_out_mfe + bp_prob(i,3);
    end
  end
end
% normalization
sum_in_win = sum_in_win/win_len;
sum_in_win_mfe = sum_in_win_mfe/n_mfe_pairs_inwin;
sum_out = sum_out/((pal_len-eb_len)/2 - win_len);
sum_out_mfe = sum_out_mfe/(n_mfe_pairs-n_mfe_pairs_inwin);
function [sum_in_win, sum_out, faulty] = ...
    base_pairing_nomfe(pal_len, bp_prob, mfe, winstart5, win_len)
% function [sum_in_win, sum_out, faulty] = ...
%    base_pairing(pal_len, bp_prob, mfe, winstart5, win_len)
% same as base_pairing but only computes these outputs (much faster)
n_pairs = size(bp_prob,1);
n_mfe_pairs = size(mfe,1);
arm5 = mfe(:,1);
arm3 = mfe(:,2);
eb_start = arm5(end)+1;
eb_end = arm3(end)-1;
eb_len = eb_end-eb_start+1;  % num nucs in end bulge
win_inds = [winstart5:winstart5+win_len-1];
if(any(intersect(win_inds,[eb_start:eb_end])) | win_inds(end)>pal_len)
   faulty = 1;
   sum_in_win = NaN;
   sum_out = NaN;
   disp('WINDOW IS ILLEGAL. RETURNING FAULTY=1.');
   disp(['window has ' num2str(length(intersect(win_inds,[eb_start:eb_end]))) ' nucs in endloop']);
   return
end
sum_in_win = 0;
sum_out = 0;
faulty = 0;
n_mfe_pairs_inwin = 0;
for i=1:n_pairs
  side5 = bp_prob(i,1);
```

```
  side3 = bp_prob(i,2);
  if(ismembc(side5,win_inds) | ismembc(side3,win_inds))
    sum_in_win = sum_in_win + bp_prob(i,3);
  else
    sum_out = sum_out + bp_prob(i,3);
  end
end
% normalization
sum_in_win = sum_in_win/win_len;
sum_out = sum_out/((pal_len-eb_len)/2 - win_len);
function [sum_in_win, sum_in_win_mfe, sum_out, sum_out_mfe, faulty] = ...
    base_pairing(pal_len, bp_prob, mfe, winstart5, win_len)
% function [sum_in_win, sum_in_win_mfe, sum_out, sum_out_mfe, faulty] = ...
%     base_pairing(pal_len, bp_prob, mfe, winstart5, win_len)
% pal_len is length of palindrom
% bp_prob is the base pairing prob matrix which has 3 cols:
% 5side index, 3side index, prob to be paired
% mfe has the pairs in the min free energy drawing
% winstart5 is the positon of the start of the window in question
% win_len is its length
% sum_in_win is the sum of the bp probs of all pairs involving a base
% in the designated window normalized by win_len
% sum_in_win_mfe is the sum of the bp probs of all pairs appearing
% in the mfe structure and involving a base in the window. this is
% normalized by the number of base pairs appearing in the mfe structure
% within the window (if only one folding possible sum_in_win_mfe=1).
% sum_out is like sum_in only all bases not in window. normalized by
% ((pal_len-eb_len)/2 - win_len).
% sum_out_mfe is like sum_in_win_mfe only for all bp not in window.
% analogous normalization.
% if window is illegal, returns faulty=1 and NAN for other values
% also note that no check is made on winstart5 and win_len being positive (which they must) - beware!
n_pairs = size(bp_prob,1);
n_mfe_pairs = size(mfe,1);
arm5 = mfe(:,1);
arm3 = mfe(:,2);
eb_start = arm5(end)+1;
eb_end = arm3(end)-1;
eb_len = eb_end-eb_start+1;  % num nucs in end bulge
win_inds = [winstart5:winstart5+win_len-1];
if(any(intersect(win_inds,[eb_start:eb_end])) | win_inds(end)>pal_len)
  faulty = 1;
  sum_in_win = NaN;
  sum_in_win_mfe = NaN;
  sum_out = NaN;
  sum_out_mfe = NaN;
  disp('WINDOW IS ILLEGAL. RETURNING FAULTY=1.');
  disp(['window has ' num2str(length(intersect(win_inds,[eb_start:eb_end]))) ' nucs in endloop']);
  return
end
```

```
sum_in_win = 0;
sum_in_win_mfe = 0;
sum_out = 0;
sum_out_mfe = 0;
faulty = 0;
n_mfe_pairs_inwin = 0;
for i=1:n_pairs
  side5 = bp_prob(i,1);
  side3 = bp_prob(i,2);
  if(ismember(side5,win_inds) | ismember(side3,win_inds))
    sum_in_win = sum_in_win + bp_prob(i,3);
    if(ismember([side5,side3],mfe,'rows'))
      sum_in_win_mfe = sum_in_win_mfe + bp_prob(i,3);
      n_mfe_pairs_inwin = n_mfe_pairs_inwin + 1;
    end
  else
    sum_out = sum_out + bp_prob(i,3);
    if(ismember([side5,side3],mfe,'rows'))
      sum_out_mfe = sum_out_mfe + bp_prob(i,3);
    end
  end
end
% normalization
sum_in_win = sum_in_win/win_len;
sum_in_win_mfe = sum_in_win_mfe/n_mfe_pairs_inwin;
sum_out = sum_out/((pal_len-eb_len)/2 - win_len);
sum_out_mfe = sum_out_mfe/(n_mfe_pairs-n_mfe_pairs_inwin);
function [sum_in_win, sum_in_win_mfe, sum_out, sum_out_mfe, faulty] = ...
    base_pairing2(pal_len, bp_prob, mfe, winstart5, win_len)
% function [sum_in_win, sum_in_win_mfe, sum_out, sum_out_mfe, faulty] = ...
%    base_pairing2(pal_len, bp_prob, mfe, winstart5, win_len)
% see base_pairing but here no normalization
n_pairs = size(bp_prob,1);
n_mfe_pairs = size(mfe,1);
arm5 = mfe(:,1);
arm3 = mfe(:,2);
eb_start = arm5(end)+1;
eb_end = arm3(end)-1;
eb_len = eb_end-eb_start+1;  % num nucs in end bulge
win_inds = [winstart5:winstart5+win_len-1];
if(any(intersect(win_inds,[eb_start:eb_end])) | win_inds(end)>pal_len)
   faulty = 1;
   sum_in_win = NaN;
   sum_in_win_mfe = NaN;
   sum_out = NaN;
   sum_out_mfe = NaN;
   disp('WINDOW IS ILLEGAL. RETURNING FAULTY=1.');
   disp(['window has ' num2str(length(intersect(win_inds,[eb_start:eb_end]))) ' nucs in endloop']);
   return
end
```

```
sum_in_win = 0;
sum_in_win_mfe = 0;
sum_out = 0;
sum_out_mfe = 0;
faulty = 0;
n_mfe_pairs_inwin = 0;
for i=1:n_pairs
  side5 = bp_prob(i,1);
  side3 = bp_prob(i,2);
  if(ismembc(side5,win_inds) | ismembc(side3,win_inds))
    sum_in_win = sum_in_win + bp_prob(i,3);
    if(ismember([side5,side3],mfe,'rows'))
      sum_in_win_mfe = sum_in_win_mfe + bp_prob(i,3);
      n_mfe_pairs_inwin = n_mfe_pairs_inwin + 1;
    end
  else
    sum_out = sum_out + bp_prob(i,3);
    if(ismember([side5,side3],mfe,'rows'))
      sum_out_mfe = sum_out_mfe + bp_prob(i,3);
    end
  end
end
function [y, df] = chi2(table)
%[y, df] = chi2(table)
%calculate chi2 valuse for m*n table, with m,n>1
n = sum(table(:));
sum2 = sum(table,1);
sum1 = sum(table,2);
% calculate the expected vals , assuming independence
ex = 0;
for i1 = 1:size(table,1);
   for i2 = 1:size(table,2)
      ex(i1,i2) = sum1(i1) *sum2(i2)/n;
   end
end
if any(sum1 == 0) | any(sum2 == 0)
   y = NaN;
else
   y = sum(sum((ex-table).^2./ex));
end
df = (size(table,1)-1)*(size(table,2)-1);
return
   function T=clusterize_prototype(seqs,maxd)
%T=clusterize_prototype(seqsd,maxd)
%clusterize by edist. all examples within cluster have edist < maxd.
%pick one prototype from each cluster
%T(i) = 1 if example i is to be used. T(i) = 0 if example i is to be ignored.
if ~all(isletter(seqs{1}))
   for i = 1:length(seqs)
      seqs{i} = int2nuc(seqs{i});
```

```
  end
end
if length(seqs{1}) > 25
  disp('sequence is longer than 25. press enter to continue');
  pause
end
nseq=length(seqs);
%maxseq=ceil(nseq/Nc);
dij=zeros((nseq-1)*nseq/2,3);
count = 0;
for i=1:nseq-1
  for j=i+1:nseq
    count = count+1;
    dij(count,:)=[editD(seqs{i},seqs{j}),i,j];
  end
end
sdij=sortrows(dij);
npair=length(sdij);
for i=1:nseq
   ss{i}=i;
end
iseq=[1:nseq];
s2g=iseq;
ng=ones(nseq,1);
Nc=0;
useg=[];
for i=1:npair
   gid=s2g(sdij(i,[2 3]));
   if ((diff(gid)~=0) & (sdij(i,1)<maxd))
      g=union(ss{gid});
      ss{gid(1)}=g;
      s2g(g)=gid(1);
      ng(g)=0;
      ng(gid(1))=length(g);
   end
end

ii=find(ng>0);
grp={ss{ii}};
T = zeros(1,length(seqs));
rand('state',121);
for i = 1:length(ii)
  s=ss{ii(i)};
  r=ceil(rand*length(s));
  T(s(r)) = 1;
end
return
function grp=clusterize_prototype1(seqs,maxd)
%T=clusterize_prototype1(seqsd,maxd)
%clusterize by edist. all examples within cluster have edist < maxd.
```

```
%grp is a cell array containing the cluster members
if ~all(isletter(seqs{1}))
  for i = 1:length(seqs)
    seqs{i} = int2nuc(seqs{i});
  end
end
if length(seqs{1}) > 25
  disp('sequence is longer than 25. press enter to continue');
  pause
end
nseq=length(seqs);
%maxseq=ceil(nseq/Nc);
dij=zeros(0.5*nseq*(nseq-1),3);
count = 0;
for i=1:nseq-1
  for j=i+1:nseq
    dij(count+1,:)=[editD(seqs{i},seqs{j}),i,j];
    count = count+1;
  end
end
sdij=sortrows(dij);
npair=length(sdij);
for i=1:nseq
   ss{i}=i;
end
iseq=[1:nseq];
s2g=iseq;
ng=ones(nseq,1);
Nc=0;
useg=[];
for i=1:npair
   gid=s2g(sdij(i,[2 3]));
   if ((diff(gid)~=0) & (sdij(i,1)<maxd))
      g=union(ss{gid});
      ss{gid(1)}=g;
      s2g(g)=gid(1);
      ng(g)=0;
      ng(gid(1))=length(g);
   end
end

ii=find(ng>0);
grp={ss{ii}};
T = zeros(1,length(seqs));
rand('state',121);
for i = 1:length(ii)
  s=ss{ii(i)};
  r=ceil(rand*length(s));
  T(s(r)) = 1;
end
```

```
return
overhang = 2;
clear set_name;
set_name = 'h104';
load_training_from_mat;
mir_win = get_win_pos_overhang_v1(anti_inds,pos,mirlen,overhang);
num_nan_wins = 0;
num_amb_wins = 0;
for i= 1:length(mir_win)
  w = mir_win{i};
  if(~isstruct(w))
    num_nan_wins = num_nan_wins+1;
  else
    if(w.ambigeous)
      num_amb_wins = num_amb_wins+1;
    end
  end
end
length(mir_win)
num_nan_wins
num_amb_wins
function create_file_for_rnastructure(seq,filename)
if(~isletter(seq(1)))
  seq=int2nuc(seq);
end
pause(1)
fid = fopen(filename,'w');
fprintf(fid,';\n1\n');
for i = 1:length(seq)
  fprintf(fid,seq(i));
end
fprintf(fid,'1\n');
fclose(fid);function h = entropy(p,base)
% function h = entropy(p,base)
% function h = entropy(p)
% computes the entropy of the distribution p in base base
% if no base is given assumes base 2
h = sum(-1*xlog2x(p));
if(nargin==2)
  h = h/log2(base);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y = xlog2x(x)
I = 1:length(x);
I0 = find(x==0);
y(I0) = 0;
I1 = setdiff(I,I0);
y(I1) = x(I1).*log2(x(I1));
function [anti_nucs,which_case] = get_anti_nucs(pos,pal_pos_bp,mfe)
```

```
% function anti_nucs = get_anti_nucs(pos,pal_pos_bp,mfe)
% pal_pos_bp is a vector of length pallen which holds the position of the nuc
% in the following format: all bp are numbered from legs by 1,2,3,...
% If a nuc is paired its pos_bp is the number of its bp. If not it is interpolated
% if the nuc is on the end loop pos_bp = 0.
% pos is the real position.
% anti_nucs gives the pos of the nuc across from pos. in some cases gives a few options.
% which_case is a string signaling the case:
% end_loop - pos sits on endloop, anti_nucs=nan in this case.
% bp - base paired
% equal_bulge
% small_bulge
% large_bulge
% non_sym_bulge
% how to determine across:
% if paired - obvious
% if on bulge and across bulge of same length, corresponding on anti bulge
% if on bulge smaller than antibulge: all options that don't cross
% if on bulge larger than antibulge and antibulge not empty: corresponding to
% all options that dont cross
% if on bulge and no bulge across: to closest bp across(if exactly in middle gives both option)
pallen = length(pal_pos_bp);
eb_start = mfe(end,1)+1;
eb_end = mfe(end,2)-1;
if(intersect([eb_start:eb_end],pos))
  %warning(['get_anti_nucs: pos sits on endloop. returning NAN. (pos = ' num2str(pos) ')']);
  anti_nucs = nan;
  which_case = 'end_loop';
  return;
end
if(pos<1 | pos>pallen)
  %warning(['get_anti_nucs: pos sits outside of pal. returning NAN. (pos = ' num2str(pos) ')']);
  anti_nucs = nan;
  which_case = 'outside pal';
  return;
end
pos_bp = pal_pos_bp(pos); % bp number of nuc in question
mod_pos_bp = mod(pos_bp,1);
if(mod_pos_bp==0) % nuc is paired
  this_pair = mfe(pos_bp,:);
  tt = find(this_pair == pos);
  anti_nucs = this_pair(setdiff([1:2],tt));
  which_case = 'bp';
  return;
end
% from here means nuc is unpaired
pos_side = 2-(pos<eb_start); % 1 for arm5, 2 for arm3.
pos_anti_side = setdiff([1:2],pos_side);
if(pos_side==1)
  my_side_inds = 1:eb_start-1;
```

```
else
  my_side_inds = eb_end+1:pallen;
end
bp_before = pos_bp - mod_pos_bp;
bp_after = bp_before + 1;
if(bp_before>0)
  num_in_my_bulge = abs(mfe(bp_after,pos_side) - mfe(bp_before,pos_side))-1;
  num_in_anti_bulge = abs(mfe(bp_after,pos_anti_side) - mfe(bp_before,pos_anti_side))-1;
else
  num_in_my_bulge = min(mfe(bp_after,pos_side)-1,abs(mfe(bp_after,pos_side)-pallen));
  num_in_anti_bulge = min(mfe(bp_after,pos_anti_side)-1,abs(mfe(bp_after,pos_anti_side)-pallen));
end
if(num_in_my_bulge == num_in_anti_bulge)
  tt = find(pal_pos_bp==pos_bp);
  anti_nucs = setdiff(tt,pos);
  which_case = 'equal_bulge';
  return;
end
my_bulge_vec = linspace(bp_before,bp_after,num_in_my_bulge+2);
my_bulge_vec = my_bulge_vec(2:end-1);
anti_bulge_vec = linspace(bp_before,bp_after,num_in_anti_bulge+2);
anti_bulge_vec = anti_bulge_vec(2:end-1);
my_place = find(my_bulge_vec==pos_bp);
if(num_in_my_bulge < num_in_anti_bulge)
  for i=1:(num_in_anti_bulge-num_in_my_bulge+1)
    tt = find(pal_pos_bp == anti_bulge_vec(my_place+i-1));
    % make sure not finding anything in my_bulge (that is look only in other side):
    anti_nucs(i) = setdiff(tt,my_side_inds);
  end
  which_case = 'small_bulge';
  return;
end
if((num_in_my_bulge > num_in_anti_bulge) & num_in_anti_bulge>0)
  anti_nucs = [];
  for i=1:(num_in_my_bulge-num_in_anti_bulge+1)
    tt = my_place-i+1;
    if(tt>0 & tt<=num_in_anti_bulge)
      ttt = find(pal_pos_bp == anti_bulge_vec(tt));
      % make sure not finding anything in my_bulge (that is look only in other side):
      anti_nucs = [anti_nucs,setdiff(ttt,my_side_inds)];
    end
  end
  which_case = 'large_bulge';
  return;
end
if(num_in_anti_bulge == 0)
  if(mod_pos_bp==0.5)
    anti_nucs(1) = mfe(bp_after,pos_anti_side);
    if(bp_before>0)
      anti_nucs(2) = mfe(bp_before,pos_anti_side);
```

```
    end
  elseif(mod_pos_bp<0.5 & bp_before>0)
    anti_nucs = mfe(bp_before,pos_anti_side);
  else
    anti_nucs = mfe(bp_after,pos_anti_side);
  end
  which_case = 'non_sym_bulge';
  return;
end
% really shouldn't be here
error('terrible mistake... aborting');

  function [energy,mfe] = get_from_ct(ct_file)
% gets the energy and mfe of first zuker fold as outputted from rnastructure
% caution: relies on the very specific format of the out file ct_file - check!
ct_file
fid = fopen(ct_file,'r');
if(fid==-1)
  keyboard
end
line = fgetl(fid);
x = findstr('ENERGY',line);
if(isempty(x))
  energy = 0;
  mfe = [];
  fclose(fid);
  return;
end
seqlen = str2num(line(1:x-1));
x = findstr('=',line);
ll = line(x+2:end);
x = findstr(' ',ll);
energy_s = ll(1:x);
energy = str2num(energy_s);
count = 0;
for i=1:seqlen
  line = fgetl(fid);
  v = str2num(line(8:end));
  across = v(3);
  if(across>0 & across<i)
    % already redundant info
    break;
  end
  if(across>0)
    count = count+1;
    mfe(count,1) = i;
    mfe(count,2) = across;
  end
end
fclose(fid);
```

```
function pos_bp = get_pos_bp(anti_inds)
% function pos_bp = get_pos_bp(anti_inds)
% pos_bp{i} is a vector of length pallen(i) holding the position of the nuc
% in the following format: all bp are numbered from legs by 1,2,3,...
% If a nuc is paired its pos_bp is the number of its bp. If not it is interpolated
% if the nuc is on the end loop pos_bp = 0
vec_flag = 0;
if(~iscell(anti_inds))
  tt{1} = anti_inds;
  anti_inds = tt;
  vec_flag = 1;
end
for i=1:length(anti_inds)
  ai = anti_inds{i};
  pallen = length(ai);
  mfe = anti_inds_to_mfe(ai);
  this_pos_bp = zeros(1,pallen);
  arm5 = mfe(:,1);
  arm3 = mfe(:,2);
  eb_start = arm5(end)+1;
  eb_end = arm3(end)-1;
  eb_len = eb_end-eb_start+1;  % num nucs in end bulge

  this_pos_bp(arm5(1)) = 1;
  this_pos_bp(arm3(1)) = 1;
  d5 = arm5(1)-1;
  for k=1:d5
    this_pos_bp(k) = k/(d5+1);
  end
  d3 = pallen-arm3(1);
  for k=1:d3
    this_pos_bp(arm3(1)+k) = (d3+1-k)/(d3+1);
  end
  for j=2:length(arm5)
    this_pos_bp(arm5(j)) = j;
    this_pos_bp(arm3(j)) = j;
    d5 = arm5(j)-arm5(j-1)-1; %how many nucs in bulge between them
    for k=1:d5
      this_pos_bp(arm5(j-1)+k) = j-1 + k/(d5+1);
    end
    d3 = arm3(j-1)-arm3(j)-1;
    for k=1:d3
      this_pos_bp(arm3(j)+k) = j-1 + (d3+1-k)/(d3+1);
    end
  end
  pos_bp{i} = this_pos_bp;
end
if(vec_flag)
  tt = pos_bp{1};
  pos_bp = tt;
```

```
end
function win_mirpos = get_win_pos_v1(mfes,anti_inds,mirpos,mirlen)
% function win_mirpos = get_win_pos(mfes,anti_inds,mirpos,mirlen)
% returns win_mirpos in index of basepair (from legs not loop).
% i.e. mfe(win_mirpos,1) is the nuc pos on the 5 arm
% for mir on arm3 returns the closest bp from its mirpos towards the legs
% for mir on arm5 returns the closest bp from its END (mirpos+mirlen-1) towards the legs
% also towards the legs
for i=1:length(mirpos)
  pos5 = mirpos(i);
  pos3 = pos5+mirlen(i)-1;
  mfe = mfes{i};
  arm5 = mfe(:,1);
  arm3 = mfe(:,2);
  eb_start = arm5(end)+1;
  eb_end = arm3(end)-1;
  eb_len = eb_end-eb_start+1;
  side5 = (pos5<eb_start);
  ai = anti_inds{i};
  is_paired = (ai~=0);
  if(side5)
    k=0;
    while(~is_paired(pos3-k))
      k=k+1;
    end
    win_mirpos(i) = find(arm5==(pos3-k));
  else
    k=0;
    while(~is_paired(pos5+k))
      k=k+1;
    end
    win_mirpos(i) = find(arm3==(pos5+k));
  end
  if(isempty(win_mirpos(i)))
    error('get_win_pos: fatal error. aborting.');
  end
end
function win_mirpos = get_win_pos_v2(mfes,anti_inds,mirpos,mirlen)
% function win_mirpos = get_win_pos(mfes,anti_inds,mirpos,mirlen)
% returns win_mirpos in index of basepair (from legs not loop).
% i.e. mfe(win_mirpos,1) is the nuc pos on the 5 arm
% for mir on arm3 returns the closest bp from its mirpos towards the loop
% for mir on arm5 returns the closest bp from its END towards the loop (mirpos+mirlen-1)
% also towards the legs
for i=1:length(mirpos)
  pos5 = mirpos(i);
  pos3 = pos5+mirlen(i)-1;
  mfe = mfes{i};
  arm5 = mfe(:,1);
  arm3 = mfe(:,2);
```

```
  eb_start = arm5(end)+1;
  eb_end = arm3(end)-1;
  eb_len = eb_end-eb_start+1;
  side5 = (pos5<eb_start);
  ai = anti_inds{i};
  is_paired = (ai~=0);
  if(side5)
    k=0;
    while(~is_paired(pos3+k))
      k=k+1;
    end
    tt = find(arm5==(pos3+k));
    if(tt)
      win_mirpos(i) = tt;
    else
      win_mirpos(i) = nan;
      disp(['mir ' num2str(i) ' intersects with loop - returning win_mirpos nan']);
    end
  else
    k=0;
    while(~is_paired(pos5-k))
      k=k+1;
    end
    tt = find(arm3==(pos5-k));
    if(tt)
      win_mirpos(i) = tt;
    else
      win_mirpos(i) = nan;
      disp(['mir ' num2str(i) ' intersects with loop - returning win_mirpos nan']);
    end
  end
  if(isempty(win_mirpos(i)))
    error('get_win_pos: fatal error. aborting.');
  end
  if(ismember(pos5,eb_start:eb_end) | ismember(pos3,eb_start:eb_end))
  end
end
function win_mirpos = get_win_pos_v3(mfes,anti_inds,mirpos,mirlen)
% function win_mirpos = get_win_pos_v3(mfes,anti_inds,mirpos,mirlen)
% returns win_mirpos in index of basepair (from legs not loop).
% i.e. mfe(win_mirpos,1) is the nuc pos on the 5 arm
% for mir on arm3 returns the closest bp from its END (mirpos+mirlen-1) towards the LOOP
% for mir on arm5 returns the closest bp from its mirpos towards the LOOP
% also towards the legs
for i=1:length(mirpos)
  pos5 = mirpos(i);
  pos3 = pos5+mirlen(i)-1;
  mfe = mfes{i};
  arm5 = mfe(:,1);
  arm3 = mfe(:,2);
```

```
  eb_start = arm5(end)+1;
  eb_end = arm3(end)-1;
  eb_len = eb_end-eb_start+1;
  side5 = (pos5<eb_start);
  ai = anti_inds{i};
  is_paired = (ai~=0);
  if(side5)
    k=0;
    while(~is_paired(pos5+k))
      k=k+1;
    end
    win_mirpos(i) = find(arm5==(pos5+k));
  else
    k=0;
    while(~is_paired(pos3-k))
      k=k+1;
    end
    win_mirpos(i) = find(arm3==(pos3-k));
  end
  if(isempty(win_mirpos(i)))
    error('get_win_pos: fatal error. aborting.');
  end
end
function get_zuker_draw_by_number(drawfile,n)
% function get_zuker_draw_by_number(drawfile,n)
% given a file of zuker draws and a number, spills on the workspace the
% zuker draw number n in the file
fid = fopen(drawfile,'r');
ind = 0;
found_flag = 0;
while (~feof(fid) & found_flag==0)
  ind = ind + 1; % index going to read now
  if(ind==n)
    found_flag==1;
    disp('.')
    for i = 1:4
      line = fgetl(fid);
      disp(line);
    end
    disp('.');
  else
    for i = 1:4
      line = fgetl(fid);
    end
  end
end
fclose(fid);function strseq = int2nuc(intseq, ncase)
%strseq = int2nuc(intseq, ncase)
%convert a sequence of '1 2 3 4' into  'A C T G'  or 'a c t g'
% ncase = uppercase | lowercase
```

```
if(isletter(intseq(1)))
   strseq = intseq;
   return;
end
if nargin == 1
   ncase = 'uppercase';
end
if strcmp(ncase,'uppercase')
   nucs = 'ACTG';
elseif strcmp(ncase,'lowercase')
   nucs = 'actg';
end
strseq = char(size(intseq));
for i = 1:length(intseq)
   strseq(i) = nucs(intseq(i));
end
return
function [yside, yprec2] = interpolate_prob_new(score, fitfile);
%[yside, yprec2] = interpolate_prob_new(score, fitfile);
% load the parameters for interpolation
load(fitfile);
%interpolate
yside = interp1(xs,ys,score,'linear');
yprec2 = interp1(xp2,yp2,score,'linear');
% extrapolate if necessary
if(min(xs)==xs(1)) % x is increasing
   yside(score<xs(1)) = ys(1);
   yprec2(score<xp2(1)) = yp2(1);
   yside(score>xs(end)) = ys(end);
   yprec2(score>xp2(end)) = yp2(end);
else % x is decreasing
   yside(score>xs(1)) = ys(1);
   yprec2(score>xp2(1)) = yp2(1);
   yside(score<xs(end)) = ys(end);
   yprec2(score<xp2(end)) = yp2(end);
end
returnfunction [yside, yprec2] = interpolate_prob_new_txt(score, fitfile);
%[yside, yprec2] = interpolate_prob_new(score, fitfile);
% fitfile is a text file
% load the parameters for interpolation
fid = fopen(fitfile,'r');
while ~feof(fid)
   line = fgetl(fid);
   if ~isstr(line), break, end;
   eval(line)
end
fclose(fid);
%interpolate
yside = interp1(xs,ys,score,'linear');
yprec2 = interp1(xp2,yp2,score,'linear');
```

```
% extrapolate if necessary
if(min(xs)==xs(1)) % x is increasing
  yside(score<xs(1)) = ys(1);
  yprec2(score<xp2(1)) = yp2(1);
  yside(score>xs(end)) = ys(end);
  yprec2(score>xp2(end)) = yp2(end);
else % x is decreasing
  yside(score>xs(1)) = ys(1);
  yprec2(score>xp2(1)) = yp2(1);
  yside(score<xs(end)) = ys(end);
  yprec2(score<xp2(end)) = yp2(end);
end
returnfunction [ry,ry_unique,mass,newx,newy,pos] = isotonic_regression(x,y)
% function [ry,ry_unique,mass,newx,neyy,pos] = isotonic_regression(x,y)
% first uniques x and attaches to it a y which the average of all y's
% attached to same x value (returns these new x and y).
% Also returns the "mass" of each point, so if a few points had the
% same x they are now lumped to one point, whose newy is the mean of
% the original ys.
% ry_unique is the regression of the "uniqued points". ry retains the
% dimensionality of the data.
% pos is such that sort(x)=newx(pos)
% (newx,ry_unique) are the new points. i.e they are sorted x's s.t. each x
% has one point y attached which is monotonous (the result of the IR).
% short short description: after running this function use as the new vectors
% newx and ry_unique
x=x(:); y=y(:);
oldx=x;
oldy=y;
if(length(x)~=length(y))
  disp('x and y must be of same length');
  return;
end
% sort the data according to x
[x,sortind]=sort(x);
y=y(sortind);
% first find avg of y's corresponding to the same x:
[x ndx pos]=unique(x);
mass=diff([0;ndx]); % uses the fact that x is sorted!!!!!
counter=1;
for t=1:length(x)
  y(t)=mean(y(counter:counter+mass(t)-1));
  counter=counter+mass(t);
end
y(length(x)+1:length(y))=[];
ry=zeros(size(x));
ry(1)=y(1);
for i=2:length(x)

  ry(i)=y(i);
```

```
  j=i;
  while(j>1)
    if(ry(j)>=ry(j-1)) break; end
    newy=sum(mass(j-1:i).*ry(j-1:i))/sum(mass(j-1:i));
    ry(j-1:i)=newy;
    j=j-1;
  end % while

end % i loop
ry_unique=ry;
ry=zeros(size(oldy));
counter=1;
for t=1:length(ry_unique)
  for j=1:mass(t)
    rytmp(counter)=ry_unique(t);
    counter=counter+1;
  end
end
ry(sortind)=rytmp;
newx=x;
newy=y;
data_dir = 'data_baseline_13_4';
%data_dir = 'data_baseline_15_5';
%data_dir = 'data_baseline_15_5\edist_above_87';
if(~exist('set_name'))
  %set_name = 'edist_above_87';
  set_name = 'h121';
end
if ~exist('randomize')
  randomize = 0;
end
if ~exist('remove_duplicate_mirs')
  remove_duplicate_mirs = 0;
end
palfile = ['c:\rosetta\' data_dir '\zuker_draw_' set_name '.txt'];
[seqs,anti_inds,bulges1,bulges2,endbulges,seq_id] = read_structure_withanti(palfile);
mirseqfile = ['c:\rosetta\' data_dir '\dicerseq_' set_name '.txt'];
[mirseqs,mirlen] = read_seq(mirseqfile);
pos = locate_dicer(mirseqs,seqs);
if randomize
  rand('state',sum(100*clock));
 disp('performing randomized permutation');
 l = randperm(length(seqs));
 bulges1 = bulges1(l);
  bulges2 = bulges2(l);
  anti_inds = anti_inds(l);
 endbulges = endbulges(l);
 mirlen = mirlen(l);
 pos = pos(l);
 seq_id = seq_id(l);
```

```
 seqs = seqs(l);
   mirseqs = mirseqs(l);
end
if remove_duplicate_mirs
   disp('removing duplicate mirs');
   D = zeros(length(seqs),1);  % list of duplicate mirs
   for i = 1:length(seqs)
      for j = i+1:length(seqs)
         if length(mirseqs{j}) == length(mirseqs{i})
            if all(mirseqs{j} == mirseqs{i})
               D(j) = 1;
               break;
            end
         end
      end
   end
   l = find(D);
   bulges1(l) = [];
   bulges2(l) = [];
   anti_inds(l) = [];
   endbulges(l) = [];
   mirlen(l) = [];
   pos(l) = [];
   seq_id(l) = [];
   seqs(l) = [];
   mirseqs(l) = [];
end
lend=mirlen; % some applications use lend and not mirlen.
data_dir = 'data_baseline_15_5';
if(~exist('set_name'))
   set_name = 'hmdc294';
end
filename =['c:\rosetta\' data_dir '\vars_' set_name]
load(filename);
mirlen = lend;if(~exist('d'))
   d = 'h121';
end
if ~exist('randomize')
   randomize = 1;
end
if ~exist('remove_duplicate_mirs')
   remove_duplicate_mirs = 1;
end
palseqfile = ['c:\rosetta\data_baseline_13_4\palseq_' d '.txt'];
[seqs,pallen] = read_seq(palseqfile);
mirseqfile = ['c:\rosetta\data_baseline_13_4\dicerseq_' d '.txt'];
[mirseqs,mirlen] = read_seq(mirseqfile);
pos = locate_dicer(mirseqs,seqs);
palmfefile = ['c:\rosetta\data_baseline_13_4\mfe_structure_' d '.txt'];
[mfes,anti_inds,bulges1,bulges2,endbulges,seq_id]= ...
```

```
  read_structure_from_mfe(palmfefile);
palbpfile = ['c:\rosetta\data_baseline_13_4\bp_prob_' d '.txt'];
[bp_probs,len] = read_bp(palbpfile);
if randomize
  rand('state',sum(100*clock));
 disp('performing randomized permutation');
 l = randperm(length(seqs));
 bulges1 = bulges1(l);
 bulges2 = bulges2(l);
 endbulges = endbulges(l);
 pallen = pallen(l);
 mirlen = mirlen(l);
 pos = pos(l);
 seq_id = seq_id(l);
 seqs = seqs(l);
  mirseqs = mirseqs(l);
  mfes = mfes(l);
  bp_probs = bp_probs(l);
  anti_inds = anti_inds(l);
end
if remove_duplicate_mirs
  disp('removing duplicate mirs');
  D = zeros(length(seqs),1);  % list of duplicate mirs
  for i = 1:length(seqs)
    for j = i+1:length(seqs)
      if length(mirseqs{j}) == length(mirseqs{i})
        if all(mirseqs{j} == mirseqs{i})
          D(j) = 1;
          break;
        end
      end
    end
  end
  l = find(D);
  bulges1(l) = [];
  bulges2(l) = [];
  endbulges(l) = [];
  mirlen(l) = [];
  pallen(l) = [];
  pos(l) = [];
  seq_id(l) = [];
  seqs(l) = [];
  mirseqs(l) = [];
  mfes(l) = [];
  bp_probs(l) = [];
  anti_inds(l) = [];
end
data_dir = 'data_baseline_29_7';
if(~exist('set_name'))
  set_name = 'h156';
```

```
end
if ~exist('randomize')
  randomize = 0;
end
if ~exist('remove_duplicate_mirs')
  remove_duplicate_mirs = 0;
end
palfile = ['c:\rosetta\' data_dir '\zuker_draw_' set_name '.txt'];
fid = fopen(palfile,'r');
[seqs,anti_inds,bulges1,bulges2,endbulges,pal_ids,energy,all_pal_ids] = ...
  read_structure_with_id_fid(fid,10000000000);
fclose(fid);
mirseqfile = ['c:\rosetta\' data_dir '\mirseq_' set_name '.txt'];
[mirseqs,mirlen,mir_ids,all_mir_ids] = read_seq_with_id(mirseqfile);
if(length(all_mir_ids)~=length(all_pal_ids) | any(all_mir_ids-all_pal_ids))
  error('ids in palfile and mirfile must match and be in same order');
end
if(length(mir_ids)~=length(pal_ids) | any(mir_ids-pal_ids))
  error('in one of the files (mir or pal) there was an illegal sequence not illegal in other file');
end
pos = locate_dicer(mirseqs,seqs);
if randomize
  rand('state',sum(100*clock));
 disp('performing randomized permutation');
 l = randperm(length(seqs));
 bulges1 = bulges1(l);
  bulges2 = bulges2(l);
  anti_inds = anti_inds(l);
 endbulges = endbulges(l);
 mirlen = mirlen(l);
 pos = pos(l);
 pal_ids = pal_ids(l);
 seqs = seqs(l);
  mirseqs = mirseqs(l);
  mir_ids = mir_ids(l);
end
if remove_duplicate_mirs
  disp('removing duplicate mirs');
  D = zeros(length(seqs),1);  % list of duplicate mirs
  for i = 1:length(seqs)
    for j = i+1:length(seqs)
      if length(mirseqs{j}) == length(mirseqs{i})
        if all(mirseqs{j} == mirseqs{i})
          D(j) = 1;
          break;
        end
      end
    end
  end
  l = find(D);
```

```
  bulges1(l) = [];
  bulges2(l) = [];
  anti_inds(l) = [];
  endbulges(l) = [];
  mirlen(l) = [];
  pos(l) = [];
  pal_ids(l) = [];
  seqs(l) = [];
  mirseqs(l) = [];
  mir_ids(l) = [];
end
lend=mirlen; % some applications use lend and not mirlen.
function pos = locate_dicer(dicer_seq,pal_seq);
%pos = locate_dicer(dicer_seq,palseq)
%get absolute position of dicer on palindrom, from the beginning of the palindrom
if length(dicer_seq) ~= length(pal_seq)
  error('different number of sequences');
end
%convert to nucleotide-format if in int format
if all(~isletter(pal_seq{1}))
  for i = 1:length(pal_seq)
    pal_seq{i} = int2nuc(pal_seq{i},'uppercase');
  end
end
if all(~isletter(dicer_seq{1}))
  for i = 1:length(dicer_seq)
    dicer_seq{i} = int2nuc(dicer_seq{i},'uppercase');
  end
end
pos = zeros(1,length(dicer_seq));
for i = 1:length(dicer_seq)
  l = findstr(dicer_seq{i}, pal_seq{i});
  if length(l) == 1
    pos(i) = l;
  else
    pos(i) = NaN;
  end
end
function y = meannan(x)
if(min(size(x))==1)
  y = mean(x(~isnan(x)));
  return;
end
y = zeros(1,size(x,2));
for i=1:size(x,2)
  v = x(:,i);
  y(i) = mean(v(~isnan(v)));
end
function seqsbp = nuc2bp(seqs,anti_inds,base_pair_basis)
%seqsbp = nuc2bp(seqs,anti_inds,base_pair_basis)
```

```
%transform to base pair representation
%for a 3 state model {AT,CG,TG} -> 1 2 3
%for a 6 state {AT,CG,TG,TA,GC,GT} -> 1 2 3 4 5 6
%also works if seqs is a vector and not a cell array, in which case returns a vector
if(~iscell(seqs))
  tt{1} = seqs;
  seqs = tt;
  tt{1} = anti_inds;
  anti_inds = tt;
  vecflag = 1;
else
  vecflag = 0;
end
map = zeros(4);
map(1,3) = 1;  %AT
map(2,4) = 2;  %CG
map(3,4) = 3;  %TG
if base_pair_basis == 3
  map = map+map';
else
  map(3,1) = 4;  %AT
  map(4,2) = 5;  %CG
  map(4,3) = 6;  %TG
end
seqsbp = cell(size(seqs));
for i = 1:length(seqs)
  seqsi = seqs{i};
  seqsbpi = zeros(size(seqsi));
  anti_indsi = anti_inds{i};

  I = find(anti_indsi ~= 0);
  for j = 1:length(I)
    ij = I(j);
    seqsbpi(ij) = map(seqsi(ij),seqsi(anti_indsi(ij)));
  end
  seqsbp{i} = seqsbpi;
end
if(vecflag)
  tt=seqsbp{1};
  seqsbp = tt;
end
return
function [intseq, fault_seq] = nuc2int(strseq);
%[intseq, fault_seq] = nuc2int(strseq)
%convert a sequence of 'A C T G' into a array of 1 2 3 4
if(~isletter(strseq(1)))
  intseq = strseq;
  fault_seq = 0;
  return;
end
```

```
intseq = zeros(size(strseq));
fault_seq = 0;
for i = 1:length(strseq)
  switch upper(strseq(i))
    case 'A' , intseq(i) = 1;
    case 'C' , intseq(i) = 2;
    case 'T' , intseq(i) = 3;
    case 'G' , intseq(i) = 4;
    otherwise , intseq = []; fault_seq = 1; break;
  end
end
function intseq = nuc2int4(strseq)
%convert a sequence of 'A C T G' into a array of 1 2 3 4
strseq = deblank(strseq);
intseq = zeros(size(strseq));
for i = 1:length(strseq)
  switch upper(strseq(i))
    case 'A' , intseq(i) = 1;
    case 'C' , intseq(i) = 2;
    case 'T' , intseq(i) = 3;
    case 'G' , intseq(i) = 4;
    otherwise , intseq(i) = [];
  end
end
function [intseq, fault_seq] = nuc2int4_new(strseq);
%[intseq, fault_seq] = nuc2int4_new(strseq)
%convert a sequence of 'A C T G' into a array of 1 2 3 4
intseq = zeros(size(strseq));
fault_seq = 0;
for i = 1:length(strseq)
  switch upper(strseq(i))
    case 'A' , intseq(i) = 1;
    case 'C' , intseq(i) = 2;
    case 'T' , intseq(i) = 3;
    case 'G' , intseq(i) = 4;
    otherwise , intseq = []; fault_seq = 1; break;
  end
end
function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid(fid,seqtot)
% function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid(fid,seqtot)
% same as read_structure_withanti_fid but reads file that have before the 4 line zuker draw
% a line giving the pal_id and a line giving the energy.
% all_pal_ids is all ids read from file, whether faulty or not
% new feature: checks that draw is not messed up and if it is gives faulty seq.
Mxplen = 250;  % maximal length of palindrom
counter = 0;
seq_no = 0;
seqs = cell(0);
```

```
bulges_nonsym= cell(0);
bulges_sym= cell(0);
endbulges = cell(0);
pal_id = zeros(0);
energy = zeros(0);
next_pal_id = str2double(fgetl(fid));
while ~feof(fid) & seq_no < seqtot
  this_pal_id = next_pal_id;
  this_energy = str2double(fgetl(fid));
  structure = char(4,250);
  i = 0;
  line = fgetl(fid);
  fault_seq_emptyline = 0;
  while((line~=-1 & isnan(str2double(line))) | isempty(line))
    if(isempty(line))
      fault_seq_emptyline = 1;
    end
    i = i+1;
    structure(i,1:length(line)) = line;
    line = fgetl(fid);
  end
  if(~feof(fid))
    next_pal_id = str2double(line);
  end
  if(i~=4)
    fault_seq_numlines = 1;
  else
    fault_seq_numlines = 0;
  end

  fault_seq_struct = 1; % guilty until proven innocent
  fault_seq_nuc = 1;
  if(fault_seq_numlines == 0 & fault_seq_emptyline==0)
    [seqi, anti_indi, bulge1i, bulge2i, endbulgei,fault_seq_struct] = get_features(structure);
    if(fault_seq_struct==0)
      % this is the old bulge1 and bulge2, now need to correct that
      bulge_nonsymi=bulge1i;
      bulge_symi=bulge2i;
      for j = 1:length(seqi)
        if(bulge_nonsymi(j))
          if(bulge_symi(max(1,j-1))) % a neighbor has a bulgesym flag on
            bulge_symi(j) = 1;
            bulge_nonsymi(j) = 0;
          end
        end
      end
      for j = length(seqi):-1:1
        if(bulge_nonsymi(j))
          if(bulge_symi(min(j+1,length(seqi)))) % a neighbor has a bulgesym flag on
            bulge_symi(j) = 1;
```

```
                bulge_nonsymi(j) = 0;
              end
            end
          end
        end
        [intseq, fault_seq_nuc] = nuc2int4_new(seqi);
      end

      if (fault_seq_struct == 0 & fault_seq_nuc == 0 & fault_seq_numlines == 0 & fault_seq_emptyline == 0)
          seq_no = seq_no + 1;
          seqs{seq_no} = intseq;
          anti_inds{seq_no} = anti_indi;
          bulges_nonsym{seq_no} = bulge_nonsymi;
          bulges_sym{seq_no} = bulge_symi;
          endbulges{seq_no} = endbulgei;
          pal_id(seq_no) = this_pal_id;
          energy(seq_no) = this_energy;
          counter = counter + 1;
          all_pal_ids(counter) = this_pal_id;
        else
          disp(['faulty seq on pal id  ' num2str(this_pal_id)])
          if(fault_seq_emptyline)
            disp(['reason is that there was an empty line in zuker']);
          elseif(fault_seq_numlines)
            disp(['reason is that there were not 4 lines in the draw']);
          elseif(fault_seq_struct)
            disp(['reason is that draw was messed has nuc in pair and bulge at the same time']);
          elseif(fault_seq_nuc)
            disp(['reason is that there was an illegal letter in the seq']);
          end
          counter = counter + 1;
          all_pal_ids(counter) = this_pal_id;
       end
    end
    return
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function [seq, anti_ind, bulge1, bulge2, endbulge, fault_seq] = get_features(structure)
    % get sequence as well as bulge structure
    fault_seq = 0;
    %upper half (5' side)
    bulge_row = 1; % the row of bulge letters
    bulge_row_opposite = 4;
    uphalf = structure(1:2,:);
    [j,k] = find(isletter(uphalf));
    max_col = max(k);
    tmpmat = zeros(2,max_col);
    count = 0;
    for col =1: max_col
      fl = find(isletter(uphalf(:,col)));
```

```
  if (length(fl)>1);
    fault_seq = 1;
    seq=nan;anti_ind=nan;bulge1=nan;bulge2=nan;endbulge=nan;
    return;
  end;
  if ~isempty(fl)
    count = count + 1;
    seq(count) = uphalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(1,col) = 0;
    else
      tmpmat(1,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
  end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
lwhalf = structure(3:4,:);
pos = length(bulge1);
while bulge1(pos) == 1
  endbulge(pos) = 1;
  bulge1(pos) = 0;
  pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(2,col) = 0;
    else
      tmpmat(2,col) = count;
    end
    bulge1(count) = 0;
```

```
      bulge2(count) = 0;
      if bulge & isletter(structure(bulge_row_opposite,col))
        bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
        bulge1(count) = 1;
      end
      endbulge(count) = 0;
    end
  end
  anti_ind = zeros(size(bulge1));
  for col=1:max_col
    if(tmpmat(1,col))
      anti_ind(tmpmat(1,col)) = tmpmat(2,col);
      anti_ind(tmpmat(2,col)) = tmpmat(1,col);
    end
  end
  return
  function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
  read_structure_with_id_fid_ce(fid,seqtot)
  % function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
  read_structure_with_id_fid_ce(fid,seqtot)
  % same as read_structure_withanti_fid but reads file that have before the 4 line zuker draw
  % a line giving the pal_id and a line giving the energy.
  % all_pal_ids is all ids read from file, whether faulty or not
  % new feature: checks that draw is not messed up and if it is gives faulty seq.
  % in this check_e version returns faulty seq also when no energy found
  Mxplen = 250;  % maximal length of palindrom
  counter = 0;
  seq_no = 0;
  seqs = cell(0);
  bulges_nonsym= cell(0);
  bulges_sym= cell(0);
  endbulges = cell(0);
  pal_id = zeros(0);
  energy = zeros(0);
  next_pal_id = str2double(fgetl(fid));
  while ~feof(fid) & seq_no < seqtot
    this_pal_id = next_pal_id;
    this_energy = str2double(fgetl(fid));
    if(isnan(this_energy))
      fault_seq_energy = 1;
    else
      fault_seq_energy = 0;
    end
    structure = char(4,250);
    i = 0;
    line = fgetl(fid);
    fault_seq_emptyline = 0;
    while((line~=-1 & isnan(str2double(line))) | isempty(line))
      if(isempty(line))
```

```
      fault_seq_emptyline = 1;
    end
    i = i+1;
    structure(i,1:length(line)) = line;
    line = fgetl(fid);
  end
  if(~feof(fid))
    next_pal_id = str2double(line);
  end
  if(i~=4)
    fault_seq_numlines = 1;
  else
    fault_seq_numlines = 0;
  end

  fault_seq_struct = 1; % guilty until proven innocent
  fault_seq_nuc = 1;
  if(fault_seq_numlines == 0 & fault_seq_emptyline==0 & fault_seq_energy==0)
    [seqi, anti_indi, bulge1i, bulge2i, endbulgei,fault_seq_struct] = get_features(structure);
    if(fault_seq_struct==0)
      % this is the old bulge1 and bulge2, now need to correct that
      bulge_nonsymi=bulge1i;
      bulge_symi=bulge2i;
      for j = 1:length(seqi)
        if(bulge_nonsymi(j))
          if(bulge_symi(max(1,j-1))) % a neighbor has a bulgesym flag on
            bulge_symi(j) = 1;
            bulge_nonsymi(j) = 0;
          end
        end
      end
      for j = length(seqi):-1:1
        if(bulge_nonsymi(j))
          if(bulge_symi(min(j+1,length(seqi)))) % a neighbor has a bulgesym flag on
            bulge_symi(j) = 1;
            bulge_nonsymi(j) = 0;
          end
        end
      end
    end
    [intseq, fault_seq_nuc] = nuc2int4_new(seqi);
  end

  if (fault_seq_struct == 0 & fault_seq_nuc == 0 & fault_seq_numlines == 0 & fault_seq_emptyline == 0 &
fault_seq_energy==0)
      seq_no = seq_no + 1;
      seqs{seq_no} = intseq;
      anti_inds{seq_no} = anti_indi;
      bulges_nonsym{seq_no} = bulge_nonsymi;
      bulges_sym{seq_no} = bulge_symi;
```

```
        endbulges{seq_no} = endbulgei;
        pal_id(seq_no) = this_pal_id;
        energy(seq_no) = this_energy;
        counter = counter + 1;
        all_pal_ids(counter) = this_pal_id;
      else
        disp(['faulty seq on pal id  ' num2str(this_pal_id)])
        if(fault_seq_energy)
          disp(['reason is that there was no energy']);
        elseif(fault_seq_emptyline)
          disp(['reason is that there was an empty line in zuker']);
        elseif(fault_seq_numlines)
          disp(['reason is that there were not 4 lines in the draw']);
        elseif(fault_seq_struct)
          disp(['reason is that draw was messed has nuc in pair and bulge at the same time']);
        elseif(fault_seq_nuc)
          disp(['reason is that there was an illegal letter in the seq']);
        end
        counter = counter + 1;
        all_pal_ids(counter) = this_pal_id;
    end
end
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [seq, anti_ind, bulge1, bulge2, endbulge, fault_seq] = get_features(structure)
% get sequence as well as bulge structure
fault_seq = 0;
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
tmpmat = zeros(2,max_col);
count = 0;
for col =1: max_col
  fl = find(isletter(uphalf(:,col)));
  if (length(fl)>1);
    fault_seq = 1;
    seq=nan;anti_ind=nan;bulge1=nan;bulge2=nan;endbulge=nan;
    return;
  end;
  if ~isempty(fl)
    count = count + 1;
    seq(count) = uphalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(1,col) = 0;
    else
```

```
      tmpmat(1,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
  end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
lwhalf = structure(3:4,:);
pos = length(bulge1);
while bulge1(pos) == 1
  endbulge(pos) = 1;
  bulge1(pos) = 0;
  pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(2,col) = 0;
    else
      tmpmat(2,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
anti_ind = zeros(size(bulge1));
for col=1:max_col
  if(tmpmat(1,col))
```

```
    anti_ind(tmpmat(1,col)) = tmpmat(2,col);
    anti_ind(tmpmat(2,col)) = tmpmat(1,col);
  end
end
return
function [xp2,yp2] = plot_errors_bins2(pos_error,score,N)
% measure the distribution of erros
if length(pos_error) ~= length(score)
  error('pos_estimated  and  score not compatible');
end
if ~exist('N')
  N = 6;
end
perc = [1:-1/N:0]*100;
thresh = prctile(score, perc);
accuracy = zeros(0);
dist1 = zeros(0); %correct size, distance = 1;
dist2 = zeros(0);
disth = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos_error);
for i = 1:length(thresh)-1
  I = find(score <= thresh(i) & score >= thresh(i+1));
  if ~isempty(I)
    count = count + 1;
    midbin(count) = mean(score(I));

    accuracy(count) = sum(pos_error(I) == 0)/length(I);
    J1 = find(abs(pos_error(I)) == 1);
    dist1(count) = length(J1)/length(I);
    J2 = find(abs(pos_error(I)) == 2);
    dist2(count) = length(J2)/length(I);
    Jh = find(abs(pos_error(I)) > 2);
    disth(count) = length(Jh)/length(I);
    fraction(count) = length(I)/N;
  else
    count = count+1;
    midbin(count) = NaN;;
    accuracy(count) = NaN;
    dist1(count) = NaN;
    dist2(count) = NaN;
    disth(count) = NaN;
    fraction(count) = NaN;
  end
end
acc1 = accuracy + dist1;
acc2 = accuracy + dist1 + dist2;
hold on
plot(midbin, acc2,'g')
```

```
plot(midbin, acc1,'r')
plot(midbin, accuracy,'b')
plot(midbin,fraction,'c')
legend('dist \leq 2', 'dist \leq 1', 'precise',2);
plot(midbin, acc2,'*g')
plot(midbin, acc1,'or')
plot(midbin, accuracy,'bd')
xlabel('bin');
%axis([min(midbin)-1 max(midbin)+1 0 1])
[ry,yp2,mass,xp2,newy,pos] = isotonic_regression(midbin,acc2);
yp2(end)
returnfunction plot_errors_perc(pos_error,score)
% measure the distribution of erros
N = 100;
perc = [1:-1/N:0]*100;
thresh = prctile(score, perc);
accuracy = zeros(0);
dist1 = zeros(0); %correct size, distance = 1;
dist2 = zeros(0);
disth = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos_error);
for i = 1:length(thresh)
   I = find(score >= thresh(i));
   if ~isempty(I)
      count = count + 1;
      accuracy(count) = sum(pos_error(I) == 0)/length(I);
      J1 = find(abs(pos_error(I)) == 1);
      dist1(count) = length(J1)/length(I);
      J2 = find(abs(pos_error(I)) == 2);
      dist2(count) = length(J2)/length(I);
      Jh = find(abs(pos_error(I)) > 2);
      disth(count) = length(Jh)/length(I);
      fraction(count) = length(I)/N;
   else
      count = count+1;
      accuracy(count) = NaN;
      dist1(count) = NaN;
      dist2(count) = NaN;
      disth(count) = NaN;
      fraction(count) = NaN;
   end
end

acc1 = accuracy + dist1;
acc2 = accuracy + dist1 + dist2;
%clf
hold on
```

```
 plot(perc, acc2,'g')
 plot(perc, acc1,'r')
 plot(perc, accuracy,'b')
 plot(perc, thresh,'c')
 legend('dist \leq 2', 'dist \leq 1', 'precise', 'threshold',2);
 xlabel('percentage');
 axis([0 100 0 1]);

 %keyboard
 %prepare result
 N = length(accuracy);
 res = [accuracy(N), acc1(N), acc2(N), acc2(round(0.2*N))]
 returnfunction y = prctile(x,p);
%PRCTILE gives the percentiles of the sample in X.
%   Y = PRCTILE(X,P) returns a value that is greater than P percent
%   of the values in X. For example, if P = 50  Y is the median of X.
%
%   P may be either a scalar or a vector. For scalar P, Y is a row
%   vector containing Pth percentile of each column of X. For vector P,
%   the ith row of Y is the P(i) percentile of each column of X.
%   Copyright (c) 1993-98 by The MathWorks, Inc.
%   $Revision: 2.6 $  $Date: 1997/11/29 01:46:27 $
[prows pcols] = size(p);
if prows ~= 1 & pcols ~= 1
    error('P must be a scalar or a vector.');
end
if any(p > 100) | any(p < 0)
    error('P must take values between 0 and 100');
end
xx = sort(x);
[m,n] = size(x);
if m==1 | n==1
    m = max(m,n);
 if m == 1,
    y = x*ones(length(p),1);
    return;
 end
    n = 1;
    q = 100*(0.5:m - 0.5)./m;
    xx = [min(x); xx(:); max(x)];
else
    q = 100*(0.5:m - 0.5)./m;
    xx = [min(x); xx; max(x)];
end
q = [0 q 100];
y = interp1(q,xx,p);
function [bps,len] = read_bp(filename);
%[bps,len] = read_bp(filename);
%reads bp file into cell array. bps{i} is a 3col matrix of the bp probs
%len(i) is the length of the ith palindrom (apears as info in the bp file)
```

```
fid = fopen(filename,'r');
if fid == -1
  error([' file ' filename '  could not be opened']);
end
seq_no = 0;
while ~feof(fid)
  pallen = str2num(fgetl(fid));
  arm5 = str2num(fgetl(fid));
  arm3 = str2num(fgetl(fid));
  p = str2num(fgetl(fid));
  seq_no = seq_no+1;
  bps{seq_no} = [arm5',arm3',p'];
  len(seq_no) = pallen;
end
fclose(fid);
return

  function [seqs,len] = read_seq(filename);
%[seqs,len] = read_seq(filename);
%reads dicer or pal sequences into cell array, in numeric format
fid = fopen(filename,'r');
if fid == -1
  error([' file ' filename '  could not be opened']);
end
id = 0;
seq_no = 0;
while ~feof(fid)
  line = fgetl(fid);
  line = deblank(line);
  [intseq, fault_seq] = nuc2int4_new(line);
  id = id + 1;
  if fault_seq == 0
    seq_no = seq_no + 1;
    seqs{seq_no} = intseq;
    len(seq_no) = length(intseq);
  else
    disp(['faulty seq on id  ' num2str(id)])
  end

  if(mod(seq_no,1000) == 0 & seq_no ~= 0)
     disp(['seq_no ' num2str(seq_no)]);
  end
end
fclose(fid);
return

  function [seqs,len, ids,all_ids] = read_seq_with_id(filename);
%[seqs,len,ids,all_ids] = read_seq_id(filename);
%reads mirr or pal sequences into cell array, in numeric format
%the input file must contain for each seq 2 lines, first is id, second is the seq
```

```
% ids holds the ids of those that were read succesfully so has same length as seqs
% all_ids is all ids encountered in the file regardless of whether were legal
fid = fopen(filename,'r');
if fid == -1
  error([' file ' filename '  could not be opened']);
end
id = 0;
seq_no = 0;
all_ids = [];
while ~feof(fid)
  this_id = str2num(fgetl(fid));
  all_ids = [all_ids,this_id];
  line = fgetl(fid);
  line = deblank(line);
  [intseq, fault_seq] = nuc2int4_new(line);
  if fault_seq == 0
    seq_no = seq_no + 1;
    seqs{seq_no} = intseq;
    len(seq_no) = length(intseq);
    ids(seq_no) = this_id;
  else
    disp(['faulty seq on id  ' num2str(id)])
  end

  if(mod(seq_no,1000) == 0 & seq_no ~= 0)
    disp(['seq_no ' num2str(seq_no)]);
  end
end
fclose(fid);
return

  function [mfes,anti_inds,bulges_nonsym,bulges_sym,endbulges,seq_id]= read_structure_from_mfe(filename);
% read rnafold structure
% seq is a cell array containing sequences (in ints)
% anti_inds holds for each nuc in the seq what is the index of the nuc across from it where the 0 means unpaired.
% bulge_nonsym is a cell array with binary strings with 1 for one sided bulge (not incl. end bulge)
% bulge_sym is similarly for 2 sided bulge
% note that any nuc ina bulge which has a bulge across gets bulge_sym even if it itself is across a -
% this is the difference from the original read_structure
% endbulge is a cell array with binary strings with 1 on the end bulge only
% the input file contains 3 lines for each paindrom. the first line is a single number indicating the pal length
% the second and third lines are the base pairs in the mfe structure
fid = fopen(filename,'r');
seq_no = 0;
mfe = cell(0);
bulges_nonsym= cell(0);
bulges_sym= cell(0);
endbulges = cell(0);
while ~feof(fid)
```

```
  pallen = str2num(fgetl(fid));
  arm5 = str2num(fgetl(fid));
  arm3 = str2num(fgetl(fid));
  seq_no = seq_no+1;

  mfes{seq_no} = [arm5',arm3'];

  ai = zeros(1,pallen);
  ai(arm5) = arm3;
  ai(arm3) = arm5;
  anti_inds{seq_no} = ai;

  ebs = zeros(1,pallen);
  eb_start = arm5(end)+1;
  eb_end = arm3(end)-1;
  ebs(eb_start:eb_end) = 1;
  endbulges{seq_no} = ebs;
  if(eb_end-eb_start+1 < 3)
    disp(['end bulge shorter than 3 nucs in seq no ' num2str(seq_no)]);
  end

  bs = zeros(1,pallen);
  bns = zeros(1,pallen);
  arm5t = [0,arm5];
  arm3t = [pallen+1,arm3];
  for i=2:length(arm5t)
    d5 = arm5t(i)-arm5t(i-1)-1;
    d3 = arm3t(i-1)-arm3t(i)-1;
    if(d5)
      if(d3)
        bs([arm5t(i-1)+1:arm5t(i)-1 , arm3t(i)+1:arm3t(i-1)-1])=1;
      else
        bns(arm5t(i-1)+1:arm5t(i)-1) = 1;
      end
    else
      if(d3)
        bns(arm3t(i)+1:arm3t(i-1)-1) = 1;
      end
    end
  end
  bulges_sym{seq_no} = bs;
  bulges_nonsym{seq_no} = bns;

end
seq_id = 1:seq_no;
fclose(fid);

  function [seqs,bulges_nonsym,bulges_sym,endbulges,seq_id] = read_structure_new(filename);
% read zuker structure
% seq is a cell array containing sequences
```

```
% bulge_nonsym is a cell array with binary strings with 1 for one sided bulge (not incl. end bulge)
% bulge_sym is similarly for 2 sided bulge
% note that any nuc ina bulge which has a bulge across gets bulge_sym even if it itself is across a -
% this is the difference from the original read_structure
% endbulge is a cell array with binary strings with 1 on the end bulge only
Mxplen = 250;  % maximal length of palindrom
fid = fopen(filename,'r');
seq_no = 0;
seqs = cell(0);
bulges_nonsym= cell(0);
bulges_sym= cell(0);
endbulges = cell(0);
seq_id = zeros(0);
id = 0;
while ~feof(fid)
  structure = char(4,250);
  for i = 1:4
    line = fgetl(fid);
    structure(i,1:length(line)) = line;
  end
  id = id +1;
  [seqi, bulge1i, bulge2i, endbulgei] = get_features(structure);
  % this is the old bulge1 and bulge2, now need to correct that
  bulge_nonsymi=bulge1i;
  bulge_symi=bulge2i;
  for j = 1:length(seqi)
    if(bulge_nonsymi(j))
      if(bulge_symi(max(1,j-1))) % a neighbor has a bulgesym flag on
        bulge_symi(j) = 1;
        bulge_nonsymi(j) = 0;
      end
    end
  end
  for j = length(seqi):-1:1
    if(bulge_nonsymi(j))
      if(bulge_symi(min(j+1,length(seqi)))) % a neighbor has a bulgesym flag on
        bulge_symi(j) = 1;
        bulge_nonsymi(j) = 0;
      end
    end
  end

  [intseq, fault_seq] = nuc2int4_new(seqi);
  if fault_seq == 0
      seq_no = seq_no + 1;
      seqs{seq_no} = intseq;
      bulges_nonsym{seq_no} = bulge_nonsymi;
      bulges_sym{seq_no} = bulge_symi;
      endbulges{seq_no} = endbulgei;
      seq_id(seq_no) = id;
```

```
    else
        disp(['faulty seq on id  ' num2str(id)])
    end
   if(mod(seq_no,1000) == 0)
      seq_no
   end
end
fclose(fid);
return
function [seq, bulge1, bulge2, endbulge] = get_features(structure)
% get sequence as well as bulge structure
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
count = 0;
for col =1: max_col
   fl = find(isletter(uphalf(:,col)));
   if ~isempty(fl)
      count = count + 1;
      seq(count) = uphalf(fl,col);
      bulge = (fl == bulge_row);
      bulge1(count) = 0;
      bulge2(count) = 0;
      if bulge & isletter(structure(bulge_row_opposite,col))
         bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
         bulge1(count) = 1;
      end
   end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
pos = length(bulge1);
while bulge1(pos) == 1
   endbulge(pos) = 1;
   bulge1(pos) = 0;
   pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
lwhalf = structure(3:4,:);
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
   fl = find(isletter(lwhalf(:,col)));
```

```
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge = (fl == bulge_row);
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
return

  function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid(fid,seqtot)
% function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid(fid,seqtot)
% same as read_structure_withanti_fid but reads file that have before the 4 line zuker draw
% a line giving the pal_id and a line giving the energy.
% all_pal_ids is all ids read from file, whether faulty or not
% new feature: checks that draw is not messed up and if it is gives faulty seq.
Mxplen = 250;  % maximal length of palindrom
counter = 0;
seq_no = 0;
seqs = cell(0);
bulges_nonsym= cell(0);
bulges_sym= cell(0);
endbulges = cell(0);
pal_id = zeros(0);
energy = zeros(0);
while ~feof(fid) & seq_no < seqtot
  this_pal_id = str2double(fgetl(fid));
  this_energy = str2double(fgetl(fid));
  structure = char(4,250);
  i = 0;
  line = fgetl(fid);
  if(isempty(line))
    line = 'emptyline';
    fault_seq_emptyline = 1;
  else
    fault_seq_emptyline = 0;
  end
  while(line(1)~='|')  % if emptyline this is always true so will go into loop
    i = i+1;
    structure(i,1:length(line)) = line;
    line = fgetl(fid);
    if(isempty(line))
```

```
      line = 'emptyline';
      fault_seq_emptyline = 1;
    end
end
if(i~=4)
   fault_seq_numlines = 1;
else
   fault_seq_numlines = 0;
end

fault_seq_struct = 1; % guilty until proven innocent
fault_seq_nuc = 1;
if(fault_seq_numlines == 0 & fault_seq_emptyline==0)
   [seqi, anti_indi, bulge1i, bulge2i, endbulgei,fault_seq_struct] = get_features(structure);
   if(fault_seq_struct==0)
      % this is the old bulge1 and bulge2, now need to correct that
      bulge_nonsymi=bulge1i;
      bulge_symi=bulge2i;
      for j = 1:length(seqi)
         if(bulge_nonsymi(j))
            if(bulge_symi(max(1,j-1))) % a neighbor has a bulgesym flag on
               bulge_symi(j) = 1;
               bulge_nonsymi(j) = 0;
            end
         end
      end
      for j = length(seqi):-1:1
         if(bulge_nonsymi(j))
            if(bulge_symi(min(j+1,length(seqi)))) % a neighbor has a bulgesym flag on
               bulge_symi(j) = 1;
               bulge_nonsymi(j) = 0;
            end
         end
      end
      [intseq, fault_seq_nuc] = nuc2int4_new(seqi);
   end
end

if (fault_seq_struct == 0 & fault_seq_nuc == 0 & fault_seq_numlines == 0 & fault_seq_emptyline == 0)
      seq_no = seq_no + 1;
      seqs{seq_no} = intseq;
      anti_inds{seq_no} = anti_indi;
      bulges_nonsym{seq_no} = bulge_nonsymi;
      bulges_sym{seq_no} = bulge_symi;
      endbulges{seq_no} = endbulgei;
      pal_id(seq_no) = this_pal_id;
      energy(seq_no) = this_energy;
      counter = counter + 1;
      all_pal_ids(counter) = this_pal_id;
   else
```

```
        disp(['faulty seq on pal id  ' num2str(this_pal_id)])
        if(fault_seq_emptyline)
          disp(['reason is that there was an empty line in zuker']);
        elseif(fault_seq_numlines)
          disp(['reason is that there were not 4 lines in the draw']);
        elseif(fault_seq_struct)
          disp(['reason is that draw was messed has nuc in pair and bulge at the same time']);
        elseif(fault_seq_nuc)
          disp(['reason is that there was an illegal letter in the seq']);
        end
        counter = counter + 1;
        all_pal_ids(counter) = this_pal_id;
    end
end
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [seq, anti_ind, bulge1, bulge2, endbulge, fault_seq] = get_features(structure)
% get sequence as well as bulge structure
fault_seq = 0;
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
tmpmat = zeros(2,max_col);
count = 0;
for col =1: max_col
   fl = find(isletter(uphalf(:,col)));
   if (length(fl)>1);
      fault_seq = 1;
      seq=nan;anti_ind=nan;bulge1=nan;bulge2=nan;endbulge=nan;
      return;
   end;
   if ~isempty(fl)
      count = count + 1;
      seq(count) = uphalf(fl,col);
      bulge = (fl == bulge_row);
      if(bulge)
         tmpmat(1,col) = 0;
      else
         tmpmat(1,col) = count;
      end
      bulge1(count) = 0;
      bulge2(count) = 0;
      if bulge & isletter(structure(bulge_row_opposite,col))
         bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
         bulge1(count) = 1;
```

```
    end
  end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
lwhalf = structure(3:4,:);
pos = length(bulge1);
while bulge1(pos) == 1
  endbulge(pos) = 1;
  bulge1(pos) = 0;
  pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(2,col) = 0;
    else
      tmpmat(2,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
anti_ind = zeros(size(bulge1));
for col=1:max_col
  if(tmpmat(1,col))
    anti_ind(tmpmat(1,col)) = tmpmat(2,col);
    anti_ind(tmpmat(2,col)) = tmpmat(1,col);
  end
end
return
function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid_ce(fid,seqtot)
% function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid_ce(fid,seqtot)
```

```
% same as read_structure_withanti_fid but reads file that have before the 4 line zuker draw
% a line giving the pal_id and a line giving the energy.
% all_pal_ids is all ids read from file, whether faulty or not
% new feature: checks that draw is not messed up and if it is gives faulty seq.
% in this check_e version returns faulty seq also when no energy found
Mxplen = 250;  % maximal length of palindrom
counter = 0;
seq_no = 0;
seqs = cell(0);
bulges_nonsym= cell(0);
bulges_sym= cell(0);
endbulges = cell(0);
pal_id = zeros(0);
energy = zeros(0);
while ~feof(fid) & seq_no < seqtot
  this_pal_id = str2double(fgetl(fid));
  this_energy = str2double(fgetl(fid));
  if(isnan(this_energy))
    fault_seq_energy = 1;
  else
    fault_seq_energy = 0;
  end
  structure = char(4,250);
  i = 0;
  line = fgetl(fid);
  if(isempty(line))
    line = 'emptyline';
    fault_seq_emptyline = 1;
  else
    fault_seq_emptyline = 0;
  end
  while(line(1)~='|')  % if emptyline this is always true so will go into loop
    i = i+1;
    structure(i,1:length(line)) = line;
    line = fgetl(fid);
    if(isempty(line))
      line = 'emptyline';
      fault_seq_emptyline = 1;
    end
  end
  if(i~=4)
    fault_seq_numlines = 1;
  else
    fault_seq_numlines = 0;
  end

  fault_seq_struct = 1; % guilty until proven innocent
  fault_seq_nuc = 1;
  if(fault_seq_numlines == 0 & fault_seq_emptyline==0 & fault_seq_energy==0)
    [seqi, anti_indi, bulge1i, bulge2i, endbulgei,fault_seq_struct] = get_features(structure);
```

```
    if(fault_seq_struct==0)
      % this is the old bulge1 and bulge2, now need to correct that
      bulge_nonsymi=bulge1i;
      bulge_symi=bulge2i;
      for j = 1:length(seqi)
        if(bulge_nonsymi(j))
          if(bulge_symi(max(1,j-1))) % a neighbor has a bulgesym flag on
            bulge_symi(j) = 1;
            bulge_nonsymi(j) = 0;
          end
        end
      end
      for j = length(seqi):-1:1
        if(bulge_nonsymi(j))
          if(bulge_symi(min(j+1,length(seqi)))) % a neighbor has a bulgesym flag on
            bulge_symi(j) = 1;
            bulge_nonsymi(j) = 0;
          end
        end
      end
      [intseq, fault_seq_nuc] = nuc2int4_new(seqi);
    end
  end

  if (fault_seq_struct == 0 & fault_seq_nuc == 0 & fault_seq_numlines == 0 & fault_seq_emptyline == 0 &
fault_seq_energy==0)
      seq_no = seq_no + 1;
      seqs{seq_no} = intseq;
      anti_inds{seq_no} = anti_indi;
      bulges_nonsym{seq_no} = bulge_nonsymi;
      bulges_sym{seq_no} = bulge_symi;
      endbulges{seq_no} = endbulgei;
      pal_id(seq_no) = this_pal_id;
      energy(seq_no) = this_energy;
      counter = counter + 1;
      all_pal_ids(counter) = this_pal_id;
    else
      disp(['faulty seq on pal id  ' num2str(this_pal_id)])
      if(fault_seq_energy)
        disp(['reason is that there was no energy']);
      elseif(fault_seq_emptyline)
        disp(['reason is that there was an empty line in zuker']);
      elseif(fault_seq_numlines)
        disp(['reason is that there were not 4 lines in the draw']);
      elseif(fault_seq_struct)
        disp(['reason is that draw was messed has nuc in pair and bulge at the same time']);
      elseif(fault_seq_nuc)
        disp(['reason is that there was an illegal letter in the seq']);
      end
      counter = counter + 1;
```

```
        all_pal_ids(counter) = this_pal_id;
    end
end
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [seq, anti_ind, bulge1, bulge2, endbulge, fault_seq] = get_features(structure)
% get sequence as well as bulge structure
fault_seq = 0;
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
tmpmat = zeros(2,max_col);
count = 0;
for col =1: max_col
   fl = find(isletter(uphalf(:,col)));
   if (length(fl)>1);
      fault_seq = 1;
      seq=nan;anti_ind=nan;bulge1=nan;bulge2=nan;endbulge=nan;
      return;
   end;
   if ~isempty(fl)
      count = count + 1;
      seq(count) = uphalf(fl,col);
      bulge = (fl == bulge_row);
      if(bulge)
         tmpmat(1,col) = 0;
      else
         tmpmat(1,col) = count;
      end
      bulge1(count) = 0;
      bulge2(count) = 0;
      if bulge & isletter(structure(bulge_row_opposite,col))
         bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
         bulge1(count) = 1;
      end
   end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
lwhalf = structure(3:4,:);
pos = length(bulge1);
while bulge1(pos) == 1
   endbulge(pos) = 1;
   bulge1(pos) = 0;
```

```
  pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(2,col) = 0;
    else
      tmpmat(2,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
anti_ind = zeros(size(bulge1));
for col=1:max_col
  if(tmpmat(1,col))
    anti_ind(tmpmat(1,col)) = tmpmat(2,col);
    anti_ind(tmpmat(2,col)) = tmpmat(1,col);
  end
end
return
function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid(fid,seqtot)
% function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid(fid,seqtot)
% same as read_structure_withanti_fid but reads file that have before the 4 line zuker draw
% a line giving the pal_id and a line giving the energy.
% all_pal_ids is all ids read from file, whether faulty or not
Mxplen = 250;  % maximal length of palindrom
counter = 0;
seq_no = 0;
seqs = cell(0);
bulges_nonsym= cell(0);
bulges_sym= cell(0);
endbulges = cell(0);
```

```
pal_id = zeros(0);
energy = zeros(0);
while ~feof(fid) & seq_no < seqtot
  this_pal_id = str2num(fgetl(fid));
  this_energy = str2num(fgetl(fid));
  structure = char(4,250);
  for i = 1:4
    line = fgetl(fid);
    structure(i,1:length(line)) = line;
  end
  [seqi, anti_indi, bulge1i, bulge2i, endbulgei] = get_features(structure);
  % this is the old bulge1 and bulge2, now need to correct that
  bulge_nonsymi=bulge1i;
  bulge_symi=bulge2i;
  for j = 1:length(seqi)
    if(bulge_nonsymi(j))
      if(bulge_symi(max(1,j-1))) % a neighbor has a bulgesym flag on
        bulge_symi(j) = 1;
        bulge_nonsymi(j) = 0;
      end
    end
  end
  for j = length(seqi):-1:1
    if(bulge_nonsymi(j))
      if(bulge_symi(min(j+1,length(seqi)))) % a neighbor has a bulgesym flag on
        bulge_symi(j) = 1;
        bulge_nonsymi(j) = 0;
      end
    end
  end

  [intseq, fault_seq] = nuc2int4_new(seqi);
  if fault_seq == 0
      seq_no = seq_no + 1;
      seqs{seq_no} = intseq;
      anti_inds{seq_no} = anti_indi;
      bulges_nonsym{seq_no} = bulge_nonsymi;
      bulges_sym{seq_no} = bulge_symi;
      endbulges{seq_no} = endbulgei;
      pal_id(seq_no) = this_pal_id;
      energy(seq_no) = this_energy;
      counter = counter + 1;
      all_pal_ids(counter) = this_pal_id;
   else
      disp(['faulty seq on pal id  ' num2str(this_pal_id)])
      counter = counter + 1;
      all_pal_ids(counter) = this_pal_id;
  end
end
return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [seq, anti_ind, bulge1, bulge2, endbulge] = get_features(structure)
% get sequence as well as bulge structure
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
tmpmat = zeros(2,max_col);
count = 0;
for col =1: max_col
  fl = find(isletter(uphalf(:,col)));
  if (length(fl)>1); keyboard;end;
  if ~isempty(fl)
    count = count + 1;
    seq(count) = uphalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(1,col) = 0;
    else
      tmpmat(1,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
  end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
lwhalf = structure(3:4,:);
pos = length(bulge1);
while bulge1(pos) == 1
  endbulge(pos) = 1;
  bulge1(pos) = 0;
  pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
```

```
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(2,col) = 0;
    else
      tmpmat(2,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
anti_ind = zeros(size(bulge1));
for col=1:max_col
  if(tmpmat(1,col))
    anti_ind(tmpmat(1,col)) = tmpmat(2,col);
    anti_ind(tmpmat(2,col)) = tmpmat(1,col);
  end
end
return
function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,seq_id] = read_structure_withanti(filename);
% read zuker structure
% seq is a cell array containing sequences (in ints)
% anti_inds holds for each nuc in the seq what is the index of the nuc across from it where the 0 means unpaired.
% bulge_nonsym is a cell array with binary strings with 1 for one sided bulge (not incl. end bulge)
% bulge_sym is similarly for 2 sided bulge
% note that any nuc ina bulge which has a bulge across gets bulge_sym even if it itself is across a -
% this is the difference from the original read_structure
% endbulge is a cell array with binary strings with 1 on the end bulge only
Mxplen = 250;  % maximal length of palindrom
fid = fopen(filename,'r');
seq_no = 0;
seqs = cell(0);
bulges_nonsym= cell(0);
bulges_sym= cell(0);
endbulges = cell(0);
seq_id = zeros(0);
id = 0;
while ~feof(fid)
  structure = char(4,250);
  for i = 1:4
    line = fgetl(fid);
    structure(i,1:length(line)) = line;
```

```
  end
  id = id +1;
  [seqi, anti_indi, bulge1i, bulge2i, endbulgei] = get_features(structure);
  % this is the old bulge1 and bulge2, now need to correct that
  bulge_nonsymi=bulge1i;
  bulge_symi=bulge2i;
  for j = 1:length(seqi)
    if(bulge_nonsymi(j))
      if(bulge_symi(max(1,j-1))) % a neighbor has a bulgesym flag on
        bulge_symi(j) = 1;
        bulge_nonsymi(j) = 0;
      end
    end
  end
  for j = length(seqi):-1:1
    if(bulge_nonsymi(j))
      if(bulge_symi(min(j+1,length(seqi)))) % a neighbor has a bulgesym flag on
        bulge_symi(j) = 1;
        bulge_nonsymi(j) = 0;
      end
    end
  end

  [intseq, fault_seq] = nuc2int4_new(seqi);
  if fault_seq == 0
      seq_no = seq_no + 1;
      seqs{seq_no} = intseq;
      anti_inds{seq_no} = anti_indi;
      bulges_nonsym{seq_no} = bulge_nonsymi;
      bulges_sym{seq_no} = bulge_symi;
      endbulges{seq_no} = endbulgei;
      seq_id(seq_no) = id;
   else
      disp(['faulty seq on id  ' num2str(id)])
   end
  if(mod(seq_no,1000) == 0)
     seq_no
  end
end
fclose(fid);
return
function [seq, anti_ind, bulge1, bulge2, endbulge] = get_features(structure)
% get sequence as well as bulge structure
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
tmpmat = zeros(2,max_col);
```

```
count = 0;
for col =1: max_col
  fl = find(isletter(uphalf(:,col)));
  if (length(fl)>1); keyboard;end;
  if ~isempty(fl)
    count = count + 1;
    seq(count) = uphalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(1,col) = 0;
    else
      tmpmat(1,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
  end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
lwhalf = structure(3:4,:);
pos = length(bulge1);
while bulge1(pos) == 1
  endbulge(pos) = 1;
  bulge1(pos) = 0;
  pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(2,col) = 0;
    else
      tmpmat(2,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
```

```
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
anti_ind = zeros(size(bulge1));
for col=1:max_col
  if(tmpmat(1,col))
    anti_ind(tmpmat(1,col)) = tmpmat(2,col);
    anti_ind(tmpmat(2,col)) = tmpmat(1,col);
  end
end

return

  function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,seq_id] = read_structure_withanti_fid(fid,seqtot);
%[seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,seq_id] = read_structure_withanti_fid(fid,seqtot);
% read zuker structure
% seq is a cell array containing sequences (in ints)
% anti_inds holds for each nuc in the seq what is the index of the nuc across from it where the 0 means unpaired.
% bulge_nonsym is a cell array with binary strings with 1 for one sided bulge (not incl. end bulge)
% bulge_sym is similarly for 2 sided bulge
% note that any nuc in a bulge which has a bulge across gets bulge_sym even if it itself is across a -
% this is the difference from the original read_structure
% endbulge is a cell array with binary strings with 1 on the end bulge only
Mxplen = 250;  % maximal length of palindrom
seq_no = 0;
seqs = cell(0);
bulges_nonsym= cell(0);
bulges_sym= cell(0);
endbulges = cell(0);
seq_id = zeros(0);
id = 0;
while ~feof(fid) & seq_no < seqtot
  structure = char(4,250);
  for i = 1:4
    line = fgetl(fid);
    structure(i,1:length(line)) = line;
  end
  id = id +1;
  [seqi, anti_indi, bulge1i, bulge2i, endbulgei] = get_features(structure);
  % this is the old bulge1 and bulge2, now need to correct that
  bulge_nonsymi=bulge1i;
  bulge_symi=bulge2i;
  for j = 1:length(seqi)
    if(bulge_nonsymi(j))
      if(bulge_symi(max(1,j-1))) % a neighbor has a bulgesym flag on
```

```
            bulge_symi(j) = 1;
            bulge_nonsymi(j) = 0;
          end
        end
      end
      for j = length(seqi):-1:1
        if(bulge_nonsymi(j))
          if(bulge_symi(min(j+1,length(seqi)))) % a neighbor has a bulgesym flag on
            bulge_symi(j) = 1;
            bulge_nonsymi(j) = 0;
          end
        end
      end

      [intseq, fault_seq] = nuc2int4_new(seqi);
      if fault_seq == 0
          seq_no = seq_no + 1;
          seqs{seq_no} = intseq;
          anti_inds{seq_no} = anti_indi;
          bulges_nonsym{seq_no} = bulge_nonsymi;
          bulges_sym{seq_no} = bulge_symi;
          endbulges{seq_no} = endbulgei;
          seq_id(seq_no) = id;
       else
          disp(['faulty seq on id  ' num2str(id)])
       end
    end
    return
    function [seq, anti_ind, bulge1, bulge2, endbulge] = get_features(structure)
    % get sequence as well as bulge structure
    %upper half (5' side)
    bulge_row = 1; % the row of bulge letters
    bulge_row_opposite = 4;
    uphalf = structure(1:2,:);
    [j,k] = find(isletter(uphalf));
    max_col = max(k);
    tmpmat = zeros(2,max_col);
    count = 0;
    for col =1: max_col
      fl = find(isletter(uphalf(:,col)));
      if (length(fl)>1); keyboard;end;
      if ~isempty(fl)
        count = count + 1;
        seq(count) = uphalf(fl,col);
        bulge = (fl == bulge_row);
        if(bulge)
          tmpmat(1,col) = 0;
        else
          tmpmat(1,col) = count;
        end
```

```
      bulge1(count) = 0;
      bulge2(count) = 0;
      if bulge & isletter(structure(bulge_row_opposite,col))
        bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
        bulge1(count) = 1;
      end
    end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
lwhalf = structure(3:4,:);
pos = length(bulge1);
while bulge1(pos) == 1
   endbulge(pos) = 1;
   bulge1(pos) = 0;
   pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
   fl = find(isletter(lwhalf(:,col)));
   if ~isempty(fl)
      count = count + 1;
      seq(count) = lwhalf(fl,col);
      bulge = (fl == bulge_row);
      if(bulge)
        tmpmat(2,col) = 0;
      else
        tmpmat(2,col) = count;
      end
      bulge1(count) = 0;
      bulge2(count) = 0;
      if bulge & isletter(structure(bulge_row_opposite,col))
        bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
        bulge1(count) = 1;
      end
      endbulge(count) = 0;
   end
end
anti_ind = zeros(size(bulge1));
for col=1:max_col
   if(tmpmat(1,col))
      anti_ind(tmpmat(1,col)) = tmpmat(2,col);
      anti_ind(tmpmat(2,col)) = tmpmat(1,col);
```

```
  end
end


return

  function y = stdnan(x)
if(min(size(x))==1)
  y = std(x(~isnan(x)));
  return;
end
y = zeros(1,size(x,2));
for i=1:size(x,2)
  v = x(:,i);
  y(i) = std(v(~isnan(v)));
end
function [sym_in_win, sym_out, faulty] = symm(pal_len,mfe,winstart5,win_len)
% function [sym_in_win, sym_out, faulty] = symm(pal_len, mfe,winstart5,win_len)
% if window is illegal, returns faulty=1 and NAN for other values
% pal_len is length of palindrom
% mfe has the pairs in the min free energy drawing
% winstart5 is the positon of the start of the window in question
% win_len is its length
% sym_in_win = number of unpaired bases in win - number in antiwin, normalized by their sum
% if win start/ends within a bulge takes in anti a proportional number of bases
% sym_out is number of unpaired on window arm - opposite arm - sym_in_win, normalized by
% total number of unpaird in both arms - those unpaird in win
% NOTE that both have a sign defined by the arm onwhich the window sits.
% also note that no check is made on winstart5 and win_len being positive (which they must) - beware!
arm5 = mfe(:,1);
arm3 = mfe(:,2);
eb_start = arm5(end)+1;
eb_end = arm3(end)-1;
eb_len = eb_end-eb_start+1;  % num nucs in end bulge
win_end = winstart5+win_len-1;
win_inds = [winstart5:win_end];
if(any(intersect(win_inds,[eb_start:eb_end])) | win_end>pal_len)
  faulty =1;
  sym_in_win = NaN;
  sym_out = NaN;
  disp('WINDOW IS ILLEGAL. RETURNING FAULTY=1.');
  disp(['window has ' num2str(length(intersect(win_inds,[eb_start:eb_end]))) ' nucs in endloop']);
  return
end
faulty = 0;
m5 = diff(arm5)-1;
m3 = -1*diff(arm3)-1;
d53 = m5-m3;
if(winstart5<eb_start)
  win_arm5 = 1; % win on arm5
```

```
else
  win_arm5 = 0;
end
% create the vector bulges from the mfe structure!
bulges = ones(1,pal_len);
bulges(arm5) = 0;
bulges(arm3) = 0;
bulged5 = sum(bulges(1:eb_start-1));
bulged3 = sum(bulges(eb_end+1:end));
bulges_win = bulges(win_inds);
inwin = sum(bulges_win);
% sum antiwin without bulges
if(win_arm5)
  tt=find(arm5-winstart5 >= 0);
  ind1=tt(1); % index in arm5 of first base in win that is paired
  antiend = arm3(ind1);
  tt=find(arm5-win_end <= 0);
  ind2=tt(end); % as ind1 but last
  antistart = arm3(ind2);
  inantiwin = sum(bulges(antistart:antiend)); % without bulges at ends of anti
  if(bulges_win(1))
    if(ind1>1)
      partonwin = (arm5(ind1)-winstart5)/(arm5(ind1)-arm5(ind1-1)-1);
      inantiwin = inantiwin + (arm3(ind1-1)-arm3(ind1)-1)*partonwin;
    else
      partonwin = (arm5(ind1)-winstart5)/(arm5(ind1)-1);
      inantiwin = inantiwin + (length(bulges)-arm3(ind1))*partonwin;
    end
  end
  if(bulges_win(end))
    partonwin = (win_end-arm5(ind2))/(arm5(ind2+1)-arm5(ind2)-1);
    inantiwin = inantiwin + (arm3(ind2)-arm3(ind2+1)-1)*partonwin;
  end
  dd = inwin-inantiwin;
  sdd = inwin+inantiwin;
  if(sdd)
    sym_in_win = dd / sdd;
  else % dd must also be 0
    sym_in_win = 0;
  end
  if(bulged5+bulged3-sdd)
    sym_out = (bulged5-bulged3-dd) / (bulged5+bulged3-sdd);
  else
    sym_out = 0;
  end
else
  tt=find(arm3-winstart5 >= 0);
  ind1=tt(end); % index in arm3 of first base in win that is paired
  antiend = arm5(ind1);
  tt=find(arm3-win_end <= 0);
```

```
  ind2=tt(1); % index in arm3 of last base in win that is paired
  antistart = arm5(ind2);
  inantiwin = sum(bulges(antistart:antiend)); % without bulges at ends of anti
  if(bulges_win(1))
    partonwin = (arm3(ind1)-winstart5)/(arm3(ind1)-arm3(ind1+1)-1);
    inantiwin = inantiwin + (arm5(ind1+1)-arm5(ind1)-1)*partonwin;
  end
  if(bulges_win(end))
    if(ind2>1)
      partonwin = (win_end-arm3(ind2))/(arm3(ind2-1)-arm3(ind2)-1);
      inantiwin = inantiwin + (arm5(ind2)-arm5(ind2-1)-1)*partonwin;
    else
      partonwin = (win_end-arm3(ind2))/(length(bulges)-arm3(ind2));
      inantiwin = inantiwin + (arm5(ind2)-1)*partonwin;
    end
  end
  dd = inwin-inantiwin;
  sdd = inwin+inantiwin;
  if(sdd)
    sym_in_win = dd / sdd;
  else % dd must also be 0
    sym_in_win = 0;
  end
  if(bulged3+bulged5-sdd)
    sym_out = (bulged3-bulged5-dd) / (bulged3+bulged5-sdd);
  else
    sym_out = 0;
  end
end
function [sym_in_win, sym_out, faulty] = symm2(pal_len,mfe,winstart5,win_len)
% function [sym_in_win, sym_out, faulty] = symm2(pal_len, mfe,winstart5,win_len)
% like symm but no normalization
arm5 = mfe(:,1);
arm3 = mfe(:,2);
eb_start = arm5(end)+1;
eb_end = arm3(end)-1;
eb_len = eb_end-eb_start+1;  % num nucs in end bulge
win_end = winstart5+win_len-1;
win_inds = [winstart5:win_end];
if(any(intersect(win_inds,[eb_start:eb_end])) | win_end>pal_len)
  faulty =1;
  sym_in_win = NaN;
  sym_out = NaN;
  disp('WINDOW IS ILLEGAL. RETURNING FAULTY=1.');
  disp(['window has ' num2str(length(intersect(win_inds,[eb_start:eb_end]))) ' nucs in endloop']);
  return
end
faulty = 0;
win_arm5 =(winstart5<eb_start);
% create the vector bulges from the mfe structure!
```

```
bulges = ones(1,pal_len);
bulges(arm5) = 0;
bulges(arm3) = 0;
bulges(eb_start:eb_end) = 0;
bulged5 = sum(bulges(1:eb_start-1));
bulged3 = sum(bulges(eb_end+1:end));
bulges_win = bulges(win_inds);
inwin = sum(bulges_win);
% sum antiwin without bulges
if(win_arm5)
  tt=find(arm5-winstart5 >= 0);
  ind1=tt(1); % index in arm5 of first base in win that is paired
  antiend = arm3(ind1);
  tt=find(arm5-win_end <= 0);
  ind2=tt(end); % as ind1 but last
  antistart = arm3(ind2);
  inantiwin = sum(bulges(antistart:antiend)); % without bulges at ends of anti
  if(bulges_win(1))
    if(ind1>1)
      partonwin = (arm5(ind1)-winstart5)/(arm5(ind1)-arm5(ind1-1)-1);
      inantiwin = inantiwin + (arm3(ind1-1)-arm3(ind1)-1)*partonwin;
    else
      partonwin = (arm5(ind1)-winstart5)/(arm5(ind1)-1);
      inantiwin = inantiwin + (length(bulges)-arm3(ind1))*partonwin;
    end
  end
  if(bulges_win(end))
    partonwin = (win_end-arm5(ind2))/(arm5(ind2+1)-arm5(ind2)-1);
    inantiwin = inantiwin + (arm3(ind2)-arm3(ind2+1)-1)*partonwin;
  end
  sym_in_win = inwin-inantiwin;
  sym_out = bulged5-bulged3-sym_in_win;
else
  tt=find(arm3-winstart5 >= 0);
  ind1=tt(end); % index in arm3 of first base in win that is paired
  antiend = arm5(ind1);
  tt=find(arm3-win_end <= 0);
  ind2=tt(1); % index in arm3 of last base in win that is paired
  antistart = arm5(ind2);
  inantiwin = sum(bulges(antistart:antiend)); % without bulges at ends of anti
  if(bulges_win(1))
    partonwin = (arm3(ind1)-winstart5)/(arm3(ind1)-arm3(ind1+1)-1);
    inantiwin = inantiwin + (arm5(ind1+1)-arm5(ind1)-1)*partonwin;
  end
  if(bulges_win(end))
    if(ind2>1)
      partonwin = (win_end-arm3(ind2))/(arm3(ind2-1)-arm3(ind2)-1);
      inantiwin = inantiwin + (arm5(ind2)-arm5(ind2-1)-1)*partonwin;
    else
      partonwin = (win_end-arm3(ind2))/(length(bulges)-arm3(ind2));
```

```
        inantiwin = inantiwin + (arm5(ind2)-1)*partonwin;
      end
    end

    sym_in_win = inwin-inantiwin;
    sym_out = bulged3-bulged5-sym_in_win;
end
function seqs = transform_format(seqs,format);
%seqs = transform_format(seqs,format);
% format is either 'int' or 'nuc'
%if format not given, toggle format from int<-> nuc
% note that assume all seqs are in same format initially
if(nargin==1)
    if all(isletter(seqs{1}))
      format = 'int';
    else
      format = 'nuc';
    end
end

if(strcmp(format,'nuc'))
 for i = 1:length(seqs)
      seqs{i} = int2nuc(seqs{i});
    end
elseif(strcmp(format,'int'))
    for i = 1:length(seqs)
      seqs{i} = nuc2int(seqs{i});
    end
else
    error('transform_format: format (if given) must be int or nuc');
end
return

function visualize_dicer_structure(seqd, filename)
%visualize_dicer_structure(seqd, filename)
% show dicer on zuker structure
%seqd is in int
Mxplen = 250;  % maximal length of palindrom
fid = fopen(filename,'r');
seq_no = 0;
seqs = cell(0);
while ~feof(fid)
    seq_no = seq_no + 1
    structure = char(4,250);
    for i = 1:4
      line = fgetl(fid);
      structure(i,1:length(line)) = line;
    end
    [seq1, bulge1, endbulge1] = get_features(structure);
    seqs{seq_no} = seq1;
```

```
pos = findstr(seqd{seq_no}, seq1)
  if ~isempty(pos)
    lend = length(seqd{seq_no});
    % search on structure for pos
    [id,jd] = dicer_on_structure(pos, lend, structure);
  else
    id = [];
    jd = [];
  end

  plot_structure(structure,id,jd);
  pause

end
return
function [id,jd] = dicer_on_structure(pos, lend, structure)
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
count = 0;
dicercount = 0;
for col =1: max_col
  fl = find(isletter(uphalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    if count >=pos & count < pos + lend
      dicercount = dicercount+1;
      id(dicercount) = fl(1);
      jd(dicercount) = col;
    end
  end
end
lwhalf = structure(3:4,:);
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    if count >=pos & count < pos + lend
      dicercount = dicercount+1;
      id(dicercount) = fl(1) + 2;
      jd(dicercount) = col;
    end
  end
end
return
function plot_structure(structure,id,jd);
yscale = 1.5;
clf
```

```
hold on
axis equal
[j,k] = find(isletter(structure));
max_col = max(k);
axis([ 0 max(75,max_col) 0 5*yscale]);
for x = 1:max_col
  for y = 1:4
    text(x,yscale*y,structure(5-y,x)); % so upper appears on top
  end
end
for k = 1:length(id);
  H = text(jd(k),yscale*(5-id(k)),structure(id(k),jd(k)));
  set(H,'color',[1 0 0]);
end
return
function [seq, bulge, endbulge] = get_features(structure)
% get sequence as well as bulge structure
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
count = 0;
for col =1: max_col
  fl = find(isletter(uphalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = uphalf(fl,col);
    bulge(count) = (fl == bulge_row);
  end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge));
pos = length(bulge);
while bulge(pos) == 1
  endbulge(pos) = 1;
  pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
lwhalf = structure(3:4,:);
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge(count) = (fl == bulge_row);
```

```
      endbulge(count) = 0;
    end
  end
  return

    function visualize_dicer_structure_gidi(seqd, filename)
  %visualize_dicer_structure(seqd, filename)
  % show dicer on zuker structure
  if(~exist('filename'))
    filename = 'c:\rosetta\data_baseline_13_4\zuker_draw_h121.txt';
  end
  Mxplen = 250;  % maximal length of palindrom
  fid = fopen(filename,'r');
  seq_no = 0;
  seqs = cell(0);
  while ~feof(fid)
    seq_no = seq_no + 1
    structure = char(4,250);
    for i = 1:4
      line = fgetl(fid);
      structure(i,1:length(line)) = line;
    end
    [seq1, bulge1, endbulge1] = get_features(structure);
    seqs{seq_no} = seq1;
    pos = findstr(seqd{seq_no}, nuc2int4(seq1));
    if ~isempty(pos)
      lend = length(seqd{seq_no});
      % search on structure for pos
      [id,jd] = dicer_on_structure(pos, lend, structure);
    else
      id = [];
      jd = [];
    end

    plot_structure(structure,id,jd);
    pause

  end
  return
  function [id,jd] = dicer_on_structure(pos, lend, structure)
  uphalf = structure(1:2,:);
  [j,k] = find(isletter(uphalf));
  max_col = max(k);
  count = 0;
  dicercount = 0;
  for col =1: max_col
    fl = find(isletter(uphalf(:,col)));
    if ~isempty(fl)
      count = count + 1;
      if count >=pos & count < pos + lend
```

```
        dicercount = dicercount+1;
        id(dicercount) = fl(1);
        jd(dicercount) = col;
      end
    end
end
lwhalf = structure(3:4,:);
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
    fl = find(isletter(lwhalf(:,col)));
    if ~isempty(fl)
      count = count + 1;
      if count >=pos & count < pos + lend
        dicercount = dicercount+1;
        id(dicercount) = fl(1) + 2;
        jd(dicercount) = col;
      end
    end
end
return
function plot_structure(structure,id,jd);
yscale = 1.5;
clf
hold on
axis equal
[j,k] = find(isletter(structure));
max_col = max(k);
axis([ 0 max(75,max_col) 0 5*yscale]);
for x = 1:max_col
    for y = 1:4
      text(x,yscale*y,structure(5-y,x)); % so upper appears on top
    end
end
for k = 1:length(id);
    H = text(jd(k),yscale*(5-id(k)),structure(id(k),jd(k)));
    set(H,'color',[1 0 0]);
end
return
function [seq, bulge, endbulge] = get_features(structure)
% get sequence as well as bulge structure
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
count = 0;
for col =1: max_col
    fl = find(isletter(uphalf(:,col)));
    if ~isempty(fl)
```

```
    count = count + 1;
    seq(count) = uphalf(fl,col);
    bulge(count) = (fl == bulge_row);
  end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge));
pos = length(bulge);
while bulge(pos) == 1
  endbulge(pos) = 1;
  pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
lwhalf = structure(3:4,:);
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge(count) = (fl == bulge_row);
    endbulge(count) = 0;
  end
end
return
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%
function run_palgrade(zuker_filename,output_filename)
load model_palgrade6_rfam3_human;
fidin = fopen(zuker_filename,'r');
fidout = fopen(output_filename,'w');
seqstot = 1000; %number of sequences to classify each loop
while ~feof(fidin)
  disp('reading structure...');

  [seqs,anti_inds,bulges1,bulges2,endbulges,pal_id,energy,all_pal_ids] = ...
    read_structure_with_id_fid_ce(fidin,seqstot);

  if(~iscell(seqs))
    tt{1} = seqs; seqs = tt; clear tt;
    tt{1} = bulges1; bulges1 = tt; clear tt;
    tt{1} = bulges2; bulges2 = tt; clear tt;
    tt{1} = endbulges; endbulges = tt; clear tt;
  end

  %take as pal only certain length from loop on each side
  if (model.pal_len_to_take_on_each_side ~= -1)
    for i = 1:length(seqs)
      s=seqs{i}; b1=bulges1{i}; b2=bulges2{i}; eb = endbulges{i};
      tt = find(eb==1);
      middle_pos = tt(1)+floor(length(tt)/2);
      ind1 = max(1,middle_pos - model.pal_len_to_take_on_each_side);
      ind2 = min(length(s),middle_pos + model.pal_len_to_take_on_each_side);
      seqs{i}= s(ind1:ind2);
      bulges1{i}=b1(ind1:ind2);
      bulges2{i}=b2(ind1:ind2);
      endbulges{i}=eb(ind1:ind2);
    end
  end

  score = get_palgrade(seqs,bulges1,bulges2,endbulges,energy,model);

  for i = 1:length(score)
    fprintf(fidout,'%d %g ',pal_id(i),score(i));
  end
end
fclose(fidin);
fclose(fidout)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%
function score = get_palgrade(seqs,bulges1,bulges2,endbulges,energy,model)
if(model.filter_by_min_complexity)
  complexity = pal_complexities(seqs,model.complexity_window_size);
  for i = 1:length(seqs);
    this_c = complexity{i};
```

```
    if(min(this_c)<model.complexity_min_min_allowed | (model.filter_by_energy & energy>model.max_energy))
      score(i) = 0;
    else
      score(i) = get_this_grade(seqs{i},bulges1{i},bulges2{i},endbulges{i},energy(i),model);
    end
  end
else
  for i = 1:length(seqs);
    if(model.filter_by_energy & energy>model.max_energy)
      score(i)=0;
    else
      score(i) = get_this_grade(seqs{i},bulges1{i},bulges2{i},endbulges{i},energy(i),model);
    end
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function score = get_this_grade(seq,b1,b2,eb,energy,model)
% normalize weights to sum of 1:
G_score = get_G_score(seq,eb,model);
nobulge_score = get_nobulge_score(b1,b2,eb,model);
nobulge_piece_score = get_nobulge_piece_score(b1,b2,eb,model);
score = G_score * nobulge_score * nobulge_piece_score;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
function score = min_max_score(min_v,max_v,dir_flag,value)
if(dir_flag == 1) % the higher the better
  score = (value - min_v)/(max_v - min_v);
elseif(dir_flag == -1) % the lower the better
  score = 1 - ((value - min_v)/(max_v - min_v));
else
  error('min_max_score: dir_flag must be 1 or -1. aborting');
end
if(score<0)
  score = 0;
end
if(score>1)
  score = 1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function s = get_G_score(seq,eb,model)
tt = find(eb);
eb_begin = tt(1);
eb_end = tt(end);
index_range = [1:eb_begin-1, eb_end+1:length(seq)];
c = zeros(1,4);
for j = index_range
  c(seq(j)) = c(seq(j)) + 1;
end
```

```
f = c/sum(c); % frequencies of letters
G_freq = f(4);
s = min_max_score(model.min_G_freq,1,1,G_freq);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function s = get_nobulge_score(b1,b2,eb,model);
eff_len = length(b1) - sum(eb); % effective length
t1 = sum(b1)/eff_len;
t2 = sum(b2)/eff_len;
f = 1- t1- t2;
s = min_max_score(model.min_nobulge,1,1,f);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function s = get_nobulge_piece_score(b1,b2,eb,model);
start_arm5 = model.num_nb_per_peice_start_arm5;
start_arm3 = model.num_nb_per_peice_start_arm3;
len = model.num_non_bulged_per_peice_len;
[n5,n3] = num_non_bulged_per_peice(b1, b2, eb, start_arm5,start_arm3,len);
m = min(n5,n3);
if(m>=model.num_non_bulged_per_peice_min)
   s = 1;
else
   s = 0;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid_ce(fid,seqtot)
% function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid_ce(fid,seqtot)
% same as read_structure_withanti_fid but reads file that have before the 4 line zuker draw
% a line giving the pal_id and a line giving the energy.
% all_pal_ids is all ids read from file, whether faulty or not
% new feature: checks that draw is not messed up and if it is gives faulty seq.
% in this check_e version returns faulty seq also when no energy found
Mxplen = 250;  % maximal length of palindrom
counter = 0;
seq_no = 0;
seqs = cell(0);
bulges_nonsym= cell(0);
bulges_sym= cell(0);
endbulges = cell(0);
pal_id = zeros(0);
energy = zeros(0);
while ~feof(fid) & seq_no < seqtot
   this_pal_id = str2double(fgetl(fid));
   this_energy = str2double(fgetl(fid));
   if(isnan(this_energy))
      fault_seq_energy = 1;
   else
```

```
  fault_seq_energy = 0;
end
structure = char(4,250);
i = 0;
line = fgetl(fid);
if(isempty(line))
  line = 'emptyline';
  fault_seq_emptyline = 1;
else
  fault_seq_emptyline = 0;
end
while(line(1)~='|')  % if emptyline this is always true so will go into loop
  i = i+1;
  structure(i,1:length(line)) = line;
  line = fgetl(fid);
  if(isempty(line))
    line = 'emptyline';
    fault_seq_emptyline = 1;
  end
end
if(i~=4)
  fault_seq_numlines = 1;
else
  fault_seq_numlines = 0;
end

fault_seq_struct = 1; % guilty until proven innocent
fault_seq_nuc = 1;
if(fault_seq_numlines == 0 & fault_seq_emptyline==0 & fault_seq_energy==0)
  [seqi, anti_indi, bulge1i, bulge2i, endbulgei,fault_seq_struct] = get_features(structure);
  if(fault_seq_struct==0)
    % this is the old bulge1 and bulge2, now need to correct that
    bulge_nonsymi=bulge1i;
    bulge_symi=bulge2i;
    for j = 1:length(seqi)
      if(bulge_nonsymi(j))
        if(bulge_symi(max(1,j-1))) % a neighbor has a bulgesym flag on
          bulge_symi(j) = 1;
          bulge_nonsymi(j) = 0;
        end
      end
    end
    for j = length(seqi):-1:1
      if(bulge_nonsymi(j))
        if(bulge_symi(min(j+1,length(seqi)))) % a neighbor has a bulgesym flag on
          bulge_symi(j) = 1;
          bulge_nonsymi(j) = 0;
        end
      end
    end
```

```
        [intseq, fault_seq_nuc] = nuc2int4_new(seqi);
      end
    end

    if (fault_seq_struct == 0 & fault_seq_nuc == 0 & fault_seq_numlines == 0 & fault_seq_emptyline == 0 &
fault_seq_energy==0)
        seq_no = seq_no + 1;
        seqs{seq_no} = intseq;
        anti_inds{seq_no} = anti_indi;
        bulges_nonsym{seq_no} = bulge_nonsymi;
        bulges_sym{seq_no} = bulge_symi;
        endbulges{seq_no} = endbulgei;
        pal_id(seq_no) = this_pal_id;
        energy(seq_no) = this_energy;
        counter = counter + 1;
        all_pal_ids(counter) = this_pal_id;
      else
        disp(['faulty seq on pal id  ' num2str(this_pal_id)])
        if(fault_seq_energy)
          disp(['reason is that there was no energy']);
        elseif(fault_seq_emptyline)
          disp(['reason is that there was an empty line in zuker']);
        elseif(fault_seq_numlines)
          disp(['reason is that there were not 4 lines in the draw']);
        elseif(fault_seq_struct)
          disp(['reason is that draw was messed has nuc in pair and bulge at the same time']);
        elseif(fault_seq_nuc)
          disp(['reason is that there was an illegal letter in the seq']);
        end
        counter = counter + 1;
        all_pal_ids(counter) = this_pal_id;
    end
end
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [seq, anti_ind, bulge1, bulge2, endbulge, fault_seq] = get_features(structure)
% get sequence as well as bulge structure
fault_seq = 0;
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
tmpmat = zeros(2,max_col);
count = 0;
for col =1: max_col
    fl = find(isletter(uphalf(:,col)));
    if (length(fl)>1);
```

```
      fault_seq = 1;
      seq=nan;anti_ind=nan;bulge1=nan;bulge2=nan;endbulge=nan;
      return;
    end;
    if ~isempty(fl)
      count = count + 1;
      seq(count) = uphalf(fl,col);
      bulge = (fl == bulge_row);
      if(bulge)
        tmpmat(1,col) = 0;
      else
        tmpmat(1,col) = count;
      end
      bulge1(count) = 0;
      bulge2(count) = 0;
      if bulge & isletter(structure(bulge_row_opposite,col))
        bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
        bulge1(count) = 1;
      end
    end
  end
  % endbulge is coded on the upper half
  % go backwards form 3' side to 5' side
  endbulge = zeros(size(bulge1));
  lwhalf = structure(3:4,:);
  pos = length(bulge1);
  while bulge1(pos) == 1
    endbulge(pos) = 1;
    bulge1(pos) = 0;
    pos = pos - 1;
  end
  %lower half
  bulge_row = 2; % 4 th line on structure is 2 line on lower half
  bulge_row_opposite = 1;
  [j,k] = find(isletter(lwhalf));
  max_col = max(k);
  for col =max_col:-1:1
    fl = find(isletter(lwhalf(:,col)));
    if ~isempty(fl)
      count = count + 1;
      seq(count) = lwhalf(fl,col);
      bulge = (fl == bulge_row);
      if(bulge)
        tmpmat(2,col) = 0;
      else
        tmpmat(2,col) = count;
      end
      bulge1(count) = 0;
      bulge2(count) = 0;
```

```
      if bulge & isletter(structure(bulge_row_opposite,col))
         bulge2(count) = 1;
      elseif bulge & ~isletter(structure(bulge_row_opposite,col))
         bulge1(count) = 1;
      end
      endbulge(count) = 0;
   end
end
anti_ind = zeros(size(bulge1));
for col=1:max_col
   if(tmpmat(1,col))
      anti_ind(tmpmat(1,col)) = tmpmat(2,col);
      anti_ind(tmpmat(2,col)) = tmpmat(1,col);
   end
end
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function c = pal_complexities(seqs,winsize,endbulges)
%c = pal_complexities(seqs,winsize,endbulges)
%c = pal_complexities(seqs,winsize)
%second version looks also at endbulge, first ignores the letters there
%c is a cell array where c{i} is a vector holding the complexity measures of
% all windows fitting in the seq of the ith pal
Ns = 4;   %number of states
if nargin == 3
   omit_endbulge = 1;
else
   omit_endbulge = 0;
end
%test if single sequence
if ~iscell(seqs)
   t = cell(1);
   t{1} = seqs;
   seqs = t;
   if omit_endbulge == 1
      t = cell(1);
      t{1} = endbulges;
      endbulges = t;
   end
   clear t
end
c = cell(0);
for i = 1:length(seqs)
   this_c = [];
   seqsi = seqs{i};
   if omit_endbulge
      eb = find(endbulges{i});
      eb_begin = eb(1);
      eb_end = eb(end);
```

```
      for j=1:eb_begin-1-(winsize-1)
         this_winseq = seqsi(j:j+winsize-1);
         this_c = [this_c,get_seq_complexity(this_winseq)];
      end
      for j=eb_end+1:length(seqsi)-(winsize-1)
         this_winseq = seqsi(j:j+winsize-1);
         this_c = [this_c,get_seq_complexity(this_winseq)];
      end
   else
      for j=1:length(seqsi)-(winsize-1)
         this_winseq = seqsi(j:j+winsize-1);
         this_c = [this_c,get_seq_complexity(this_winseq)];
      end
   end
   c{i} = this_c;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%
function  c = get_seq_complexity(seq)
p = zeros(1,4);
for j=1:length(seq)
   p(seq(j)) = p(seq(j)) + 1;
end
p = p/sum(p); % letter freq in this seq
c = entropy(p); % complexity is simply the entropy of the seq in the window

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%
function [n5,n3] = ...
   num_non_bulged_per_peice(bulges1, bulges2, endbulges, piece_start_arm5, piece_start_arm3 ,piece_len)
if(~iscell(bulges1))
   tt{1} = bulges1;
   bulges1 = tt;
   clear tt;
   tt{1} = bulges2;
   bulges2 = tt;
   clear tt;

tt{1} = endbulges;
   endbulges = tt;
   clear tt;
end
numseqs = length(bulges1);
for i=1:numseqs
   eb=endbulges{i}; b1 = bulges1{i}; b2 = bulges2{i};
   tt=find(eb==1);
   loop_start=tt(1);
   loop_end=tt(end);
   nb = 1-max(b1,b2);
   s5 = max(loop_start-piece_start_arm5,1);
```

```
   e5 = max(loop_start-(piece_start_arm5+piece_len-1),1);
   n5(i) = sum(nb(s5:-1:e5));
   s3 = min(loop_end+piece_start_arm3,length(nb));
   e3 = min(loop_end+piece_start_arm3+piece_len-1,length(nb));
   n3(i) = sum(nb(s3:e3));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%
%bulge_distribution
function p = bulge_distribution(bulges1, bulges2, endbulges)
% p(i,1) - freq of bulge of type 1.
% p(i,2) - freq of bulge of type 2.
% p(i,3) - freq of no bulge (sum is 1).
% does not take into account the endbulge
if(~iscell(bulges1))
   tt{1} = bulges1;
   bulges1 = tt;
   clear tt;
   tt{1} = bulges2;
   bulges2 = tt;
   clear tt;

tt{1} = endbulges;
   endbulges = tt;
   clear tt;
end
Ns = 3; % 3 states:  1 (bulge1) 2 (bulge2) 3 (no bulge)
n = length(bulges1);
p = zeros(n,Ns);
for i = 1:n
   eff_len = length(bulges1{i}) - sum(endbulges{i}); % effective length
   p(i,1) = sum(bulges1{i})/eff_len;
   p(i,2) = sum(bulges2{i})/eff_len;
   p(i,3) = 1- p(i,1)- p(i,2);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%
%entropy
function h = entropy(p,base)
% function h = entropy(p,base)
% function h = entropy(p)
% computes the entropy of the distribution p in base base
% if no base is given assumes base 2
h = sum(-1*xlog2x(p));
if(nargin==2)
   h = h/log2(base);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function y = xlog2x(x)
```

```
I = 1:length(x);
I0 = find(x==0);
y(I0) = 0;
I1 = setdiff(I,I0);
y(I1) = x(I1).*log2(x(I1));
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%nuc2int4_new
function [intseq, fault_seq] = nuc2int4_new(strseq);
%[intseq, fault_seq] = nuc2int4_new(strseq)
%convert a sequence of 'A C T G' into a array of 1 2 3 4
intseq = zeros(size(strseq));
fault_seq = 0;
for i = 1:length(strseq)
   switch upper(strseq(i))
      case 'A' , intseq(i) = 1;
      case 'C' , intseq(i) = 2;
      case 'T' , intseq(i) = 3;
      case 'G' , intseq(i) = 4;
      otherwise , intseq = []; fault_seq = 1; break;
   end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%params6
% general
model_params.data_used_for_known_pals = 'rfam3 humans';
model_params.filter_by_energy = 1;
model_params.max_energy = -15;
% doesn't look at whole pal. Rather takes the middle of the end loop and from
% each side of it pal_len_to_take_on_each_side. If -1 takes whole pal.
model_params.pal_len_to_take_on_each_side = 41;
% minimal frequence of G accepted. score grows linear above that.
model_params.min_G_freq = 0.16;
% bulge stuff. uses ratio of no bulges (like ratio of paired)
% doesnt take into account the endbulge
% 0-1 linearly above a threshold
model_params.min_nobulge = 0.7;
% bulge count in a certain position from loop (see README)
model_params.num_nb_per_peice_start_arm5 = 18;
model_params.num_nb_per_peice_start_arm3 = 18;
model_params.num_non_bulged_per_peice_len = 5;
model_params.num_non_bulged_per_peice_min = 4;
% complexity stuff. if filter_by_min_complexity=1 filters by complexity.
% runs on windows of size complexity_window_size and computes the
% entropy in that window. Then looks for each pal at the minimal entropy
% in all of its windows. If that is less than complexity_min_min_allowed
% gives a score of 0 to that pal. else goes on as usual.
model_params.filter_by_min_complexity = 1;
model_params.complexity_window_size = 10;
model_params.complexity_min_min_allowed = 0.7;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%save_model
% homology = nan is considered 0 for histogram.
% also scores of edist and 2stage nan is taken as 0
paramfile = 'params6';
eval(paramfile);
model = model_params;
if(1)
set_name = 'human_pals_rfam3';
fid_k = fopen(['c:\rosetta_alg\data_baseline_1_3_04\' set_name '_zuker_draw.txt'],'r')
[seqs_k,anti_inds_k,bulges1_k,bulges2_k,endbulges_k,pal_id_k,energy_k,all_pal_ids_k] = ...
  read_structure_with_id_fid_ce(fid_k,1000);
fclose(fid_k);
if(length(pal_id_k)~=length(all_pal_ids_k))
  error('in training data do not allow faulty seqs, take out of there');
end
if (model_params.pal_len_to_take_on_each_side ~= -1)
  for i = 1:length(seqs_k)
    s=seqs_k{i}; b1=bulges1_k{i}; b2=bulges2_k{i}; eb = endbulges_k{i};
    tt = find(eb==1);
    middle_pos = tt(1)+floor(length(tt)/2);
    ind1 = max(1,middle_pos - model.pal_len_to_take_on_each_side);
    ind2 = min(length(s),middle_pos + model.pal_len_to_take_on_each_side);
    seqs_k{i}= s(ind1:ind2);
    bulges1_k{i}=b1(ind1:ind2);
    bulges2_k{i}=b2(ind1:ind2);
    endbulges_k{i}=eb(ind1:ind2);
  end
end
fid_1000= fopen('c:\rosetta_alg\data_baseline_1_3_04\chr_14_15_rand1583_pals_zuker_draw.txt','r');
[seqs_1000,anti_inds_1000,bulges1_1000,bulges2_1000,endbulges_1000,pal_id_1000,energy_1000,all_pal_ids_100
0] = ...
  read_structure_with_id_fid_ce(fid_1000,1000);
fclose(fid_1000);
if (model_params.pal_len_to_take_on_each_side ~= -1)
  for i = 1:length(seqs_1000)
    s=seqs_1000{i}; b1=bulges1_1000{i}; b2=bulges2_1000{i}; eb = endbulges_1000{i};
    tt = find(eb==1);
    middle_pos = tt(1)+floor(length(tt)/2);
    ind1 = max(1,middle_pos - model.pal_len_to_take_on_each_side);
    ind2 = min(length(s),middle_pos + model.pal_len_to_take_on_each_side);
    seqs_1000{i}= s(ind1:ind2);
    bulges1_1000{i}=b1(ind1:ind2);
    bulges2_1000{i}=b2(ind1:ind2);
    endbulges_1000{i}=eb(ind1:ind2);
  end
end
end %if 0/1
save model_palgrade6_rfam3_human model
[score_known] = get_palgrade(seqs_k,bulges1_k,bulges2_k,...
  endbulges_k,energy_k,model);
```

```
[score_1000] = get_palgrade(seqs_1000,bulges1_1000,bulges2_1000,...
   endbulges_1000,energy_1000,model);
hist_vec = [0:0.01:1];
cos1 = 0.5;
cos2 = 0.8;
[n_known,x] = hist(score_known,hist_vec);
n_known_norm = n_known/sum(n_known);
[n_1000,x] = hist(score_1000,hist_vec);
n_1000_norm = n_1000/sum(n_1000);
figure;
plot(x,n_known_norm,'b-o',x,n_1000_norm,'r-*','linewidth',2);
axis_vec = [min(hist_vec), max(hist_vec), 0 ,1];
axis_vec = [min(hist_vec), 0.2, 0 ,1];
axis(axis_vec);
legend('known','bg');
print -djpeg mfold_known_background
```

```
function mfe = anti_inds_to_mfe(anti_inds)
% anti_inds holds for each nuc in the seq what is the index of
% the nuc across from it where the 0 means unpaired (this is returned by read_structure_withanti).
% returns mfe which is the structure in the format of rnafold, i.e. only base pairs:
% mfe is a 2 col matrix, the first being the bases on arm5 which are paired and the second
% their corresponding pairs
if(~iscell(anti_inds))
  mfe = get_mfe(anti_inds);
  return;
end
for i=1:length(anti_inds)
  mfe{i} = get_mfe(anti_inds{i});
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%
function mfe = get_mfe(ai)
bps=0;
for i=1:length(ai)
  if(ai(i))
    if(i>ai(i))
      return
    end
    bps = bps+1;
    mfe(bps,1) = i;
    mfe(bps,2) = ai(i);
  end
end
mfold_cv_proto;
score(examples) = win_score(examples).*pos_score(examples);
%score(examples) = pos_score(examples);
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est(examples),score(examples),mirpos(examples),endbulges(examples));
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] =
analyse_errors_bins2(pos_est(examples),score(examples),mirpos(examples),endbulges(examples),num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
mfold_cv_proto;
%score(examples) = win_score(examples).*pos_score(examples);
score(examples) = win_score(examples);
%score(examples) = pos_score(examples);
for i=1:length(mirpos)
  mfe = mfes{i};
  pos_est_arm5(i) = max(1,(mfe(win_pos_est(i),1) - model.win_len + 1));
```

```
  pos_est_arm3(i) = mfe(win_pos_est(i),2);
  d5 = abs(pos_est_arm5(i)-mirpos(i));
  d3 = abs(pos_est_arm3(i)-mirpos(i));
  pos_error(i) = min(d5,d3);
  if(d3<d5)
    pos_est_side_known(i) = pos_est_arm3(i);
  else
    pos_est_side_known(i) = pos_est_arm5(i);
  end
end
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est(examples),score(examples),mirpos(examples),endbulges(examples));
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] =
analyse_errors_bins2(pos_est(examples),score(examples),mirpos(examples),endbulges(examples),num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
figure
subplot(2,1,1)
res =
analyse_errors_perc(pos_est_side_known(examples),score(examples),mirpos(examples),endbulges(examples));
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] =
analyse_errors_bins2(pos_est_side_known(examples),score(examples),mirpos(examples),endbulges(examples),num
_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
save_mfold_data = 1;
filename = 'mfold_rand5_rundata.mat';
randstate=5;
mfold_cv_random;
%score = win_score.*pos_score;
score = win_score;
%score = pos_score;
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est,score,mirpos,endbulges);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
```

```
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] = analyse_errors_bins2(pos_est,score,mirpos,endbulges,num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
if(save_mfold_data)
  eval(['save ' filename]);
end
mfold_cv_random;
%score = win_score.*pos_score;
score = win_score;
%score = pos_score;
for i=1:length(mirpos)
  mfe = mfes{i};
  pos_est_arm5(i) = max(1,(mfe(win_pos_est(i),1) - model.win_len + 1));
  pos_est_arm3(i) = mfe(win_pos_est(i),2);
  d5 = abs(pos_est_arm5(i)-mirpos(i));
  d3 = abs(pos_est_arm3(i)-mirpos(i));
  pos_error(i) = min(d5,d3);
  if(d3<d5)
    pos_est_side_known(i) = pos_est_arm3(i);
  else
    pos_est_side_known(i) = pos_est_arm5(i);
  end
end
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est,score,mirpos,endbulges);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] = analyse_errors_bins2(pos_est,score,mirpos,endbulges,num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est_side_known,score,mirpos,endbulges);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] = analyse_errors_bins2(pos_est_side_known,score,mirpos,endbulges,num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
```

```
legend('off')
mfold_cv_testwin_proto;
% chooses the correct side to only test win prediction and not side prediction
for i=1:length(examples)
  ind = examples(i);
  mfe = mfes{ind};
  pos_est_arm5 = max(1,(mfe(win_pos_est(i),1) - model.win_len + 1));
  pos_est_arm3 = mfe(win_pos_est(ind),2);
  d5 = abs(pos_est_arm5-mirpos(ind));
  d3 = abs(pos_est_arm3-mirpos(ind));
  pos_error(ind) = min(d5,d3);
  if(d3<d5)
    pos_est(ind) = pos_est_arm3;
  else
    pos_est(ind) = pos_est_arm5;
  end
end
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est(examples),win_score(examples),mirpos(examples),endbulges(examples));
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] =
analyse_errors_bins2(pos_est(examples),win_score(examples),mirpos(examples),endbulges(examples),num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
mfold_cv_testwin_random;
% chooses the correct side to only test win prediction and not side prediction
for i=1:length(mirpos)
  mfe = mfes{i};
  pos_est_arm5 = max(1,(mfe(win_pos_est(i),1) - model.win_len + 1));
  pos_est_arm3 = mfe(win_pos_est(i),2);
  d5 = abs(pos_est_arm5-mirpos(i));
  d3 = abs(pos_est_arm3-mirpos(i));
  pos_error(i) = min(d5,d3);
  if(d3<d5)
    pos_est(i) = pos_est_arm3;
  else
    pos_est(i) = pos_est_arm5;
  end
end
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est,win_score,mirpos,endbulges);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
```

```
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] = analyse_errors_bins2(pos_est,win_score,mirpos,endbulges,num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
function model = bayes_learn_pos_given_win(seqs,anti_inds,bulges1,bulges2,endbulges,pos,mirlen,model)
%model is a struct.
% mfes{i} holds the structure in the basepair notation
mfes = anti_inds_to_mfe(anti_inds);
% win_pos(i) is the position of the window corresponding to mir i
win_pos = get_win_pos_v1(mfes,anti_inds,pos,mirlen);
possible_positions = get_possible_positions(model,mfes,endbulges,win_pos);
% for each seq hold the mirposition and all possible positions that are not mirpos
for i=1:length(pos)
  mirpos(i) = pos(i);
  nonmirpos{i} = setdiff(possible_positions{i},mirpos(i));
end
[upper_mean_dist,upper_std_dist,lower_mean_dist,lower_std_dist] = loopdist_model(mirpos,endbulges);
model.pos_upper_mean_dist = upper_mean_dist;
model.pos_upper_std_dist = upper_std_dist;
model.pos_lower_mean_dist = lower_mean_dist;
model.pos_lower_std_dist = lower_std_dist;
[p1_nuc_mir,p2_nuc_mir]= nucleotide_pos_model_list(model,seqs,mirpos);
[p1_nuc_nonmir,p2_nuc_nonmir]= nucleotide_pos_model_list(model,seqs,nonmirpos);
model.pos_p1_nuc_mir = p1_nuc_mir;
model.pos_p2_nuc_mir = p2_nuc_mir;
model.pos_p1_nuc_nonmir = p1_nuc_nonmir;
model.pos_p2_nuc_nonmir = p2_nuc_nonmir;
[pb1_mir,pb2_mir,pbtot_mir] = pos_bulge_pos_model_list(model,bulges1,bulges2,mirpos);
[pb1_nonmir,pb2_nonmir,pbtot_nonmir] = pos_bulge_pos_model_list(model,bulges1,bulges2,nonmirpos);
model.pos_pb1_mir = pb1_mir;
model.pos_pb1_nonmir = pb1_nonmir;
model.pos_pb2_mir = pb2_mir;
model.pos_pb2_nonmir = pb2_nonmir;
model.pos_pbtot_mir = pbtot_mir;
model.pos_pbtot_nonmir = pbtot_nonmir;
p_bp_mir = pos_base_pair_model_list(model,seqs,anti_inds,mirpos);
p_bp_nonmir = pos_base_pair_model_list(model,seqs,anti_inds,nonmirpos);
model.p_bp_mir = p_bp_mir;
model.p_bp_nonmir = p_bp_nonmir;
function model = bayes_learn_win(seqs,anti_inds,bulges1,bulges2,endbulges,pos,mirlen,model)
%model_params is a struct.
% mfes{i} holds the structure in the basepair notation
mfes = anti_inds_to_mfe(anti_inds);
% win_pos(i) is the position of the window corresponding to mir i
win_pos = get_win_pos_v1(mfes,anti_inds,pos,mirlen);
% for each seq hold the mirposition and all possible positions that are not mirpos
for i=1:length(pos)
```

```
  mirwin(i) = win_pos(i);
  n_bps = size(mfes{i},1);
  nonmirwin{i} = setdiff([model.min_win_bp:n_bps],mirwin(i));
end
[mean_loopdist,std_loopdist] = loopdist_bp_model_normal(win_pos,mfes);
model.mean_loopdist_bp = mean_loopdist;
model.std_loopdist_bp = std_loopdist;
[win_num_bps_mir_vals,win_num_bps_mir_ps] = num_bps_model_hist_list(mfes,anti_inds,model,mirwin);
[win_num_bps_nonmir_vals,win_num_bps_nonmir_ps] = num_bps_model_hist_list(mfes,anti_inds,model,nonmirwin);
model.win_num_bps_mir_vals = win_num_bps_mir_vals;
model.win_num_bps_mir_ps = win_num_bps_mir_ps;
model.win_num_bps_nonmir_vals = win_num_bps_nonmir_vals;
model.win_num_bps_nonmir_ps = win_num_bps_nonmir_ps;
[win_sym_mir_vals,win_sym_mir_ps] = win_sym_model_list(mfes,anti_inds,model,mirwin);
[win_sym_nonmir_vals,win_sym_nonmir_ps] = win_sym_model_list(mfes,anti_inds,model,nonmirwin);
model.win_sym_mir_vals = win_sym_mir_vals;
model.win_sym_mir_ps = win_sym_mir_ps;
model.win_sym_nonmir_vals = win_sym_nonmir_vals;
model.win_sym_nonmir_ps = win_sym_nonmir_ps;
[pb_arm5_mir,pb_arm3_mir,pb1_arm5_mir,pb1_arm3_mir,pb2_arm5_mir,pb2_arm3_mir]...
  = win_bulge_pos_model_list(mfes,bulges1,bulges2,model,mirwin);
[pb_arm5_nonmir,pb_arm3_nonmir,pb1_arm5_nonmir,pb1_arm3_nonmir,pb2_arm5_nonmir,pb2_arm3_nonmir]...
  = win_bulge_pos_model_list(mfes,bulges1,bulges2,model,nonmirwin);
model.win_bulge_posit_arm5_mir = pb_arm5_mir;
model.win_bulge_posit_arm3_mir = pb_arm3_mir;
model.win_bulge1_posit_arm5_mir = pb1_arm5_mir;
model.win_bulge1_posit_arm3_mir = pb1_arm3_mir;
model.win_bulge2_posit_arm5_mir = pb2_arm5_mir;
model.win_bulge2_posit_arm3_mir = pb2_arm3_mir;
model.win_bulge_posit_arm5_nonmir = pb_arm5_nonmir;
model.win_bulge_posit_arm3_nonmir = pb_arm3_nonmir;
model.win_bulge1_posit_arm5_nonmir = pb1_arm5_nonmir;
model.win_bulge1_posit_arm3_nonmir = pb1_arm3_nonmir;
model.win_bulge2_posit_arm5_nonmir = pb2_arm5_nonmir;
model.win_bulge2_posit_arm3_nonmir = pb2_arm3_nonmir;
[win_p_bp_arm5_mir,win_p_bp_arm3_mir] = ...
  win_base_pair_model_list(mfes,anti_inds,seqs,model,mirwin);
[win_p_bp_arm5_nonmir,win_p_bp_arm3_nonmir] =...
  win_base_pair_model_list(mfes,anti_inds,seqs,model,nonmirwin);
model.win_base_pair_arm5_mir = win_p_bp_arm5_mir;
model.win_base_pair_arm3_mir = win_p_bp_arm3_mir;
model.win_base_pair_arm5_nonmir = win_p_bp_arm5_nonmir;
model.win_base_pair_arm3_nonmir = win_p_bp_arm3_nonmir;
[p1_5_mir,p2_5_mir,p1_3_mir,p2_3_mir] = win_nuc_positional_model_list(seqs,mfes,model,mirwin);
[p1_5_nonmir,p2_5_nonmir,p1_3_nonmir,p2_3_nonmir] = ...
  win_nuc_positional_model_list(seqs,mfes,model,nonmirwin);
model.win_nuc_pos_p1_5_mir = p1_5_mir;
model.win_nuc_pos_p2_5_mir = p2_5_mir;
model.win_nuc_pos_p1_3_mir = p1_3_mir;
model.win_nuc_pos_p2_3_mir = p2_3_mir;
```

```
model.win_nuc_pos_p1_5_nonmir = p1_5_nonmir;
model.win_nuc_pos_p2_5_nonmir = p2_5_nonmir;
model.win_nuc_pos_p1_3_nonmir = p1_3_nonmir;
model.win_nuc_pos_p2_3_nonmir = p2_3_nonmir;
return
function [pos,score] = bayes_predict_pos_given_win(seqs,win_pos,anti_inds,bulges1,bulges2,endbulges,model)
mfes = anti_inds_to_mfe(anti_inds);
for i = 1:length(seqs)
   %disp(num2str(i));
   [posi, scorei] =
bayes_predict_side_i(model,seqs{i},win_pos(i),mfes{i},anti_inds{i},bulges1{i},bulges2{i},endbulges{i});
   pos(i) = posi;
   score(i) = scorei;
end
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [posi, scorei] = bayes_predict_side_i(model,seqsi,wp,mfei,ai,bulges1i,bulges2i,endbulgesi)
pl = get_possible_positions(model,mfei,endbulgesi,wp);
pos_list = pl{1};
p_loopdist = loopdist_prob(pos_list,model,endbulgesi);
[p_pos_nuc_mir,p_pos_nuc_nonmir] = nuc_pos_prob(pos_list,model,seqsi);
[p_pos_bulge_mir,p_pos_bulge_nonmir] = bulge_pos_prob(pos_list,model,bulges1i,bulges2i);
[p_base_pair_mir,p_base_pair_nonmir] = base_pair_prob(pos_list,model,seqsi,ai);
p_mir = ones(size(pos_list));
p_nonmir = ones(size(pos_list));
if(model.pos_use_loopdist)
   p_mir = p_mir.*p_loopdist;
   p_nonmir = p_nonmir.*(1-p_loopdist);
end
if(model.pos_use_pos_nuc)
   p_mir = p_mir.*p_pos_nuc_mir;
   p_nonmir = p_nonmir.*p_pos_nuc_nonmir;
end
if(model.pos_use_pos_bulge)
   p_mir = p_mir.*p_pos_bulge_mir;
   p_nonmir = p_nonmir.*p_pos_bulge_nonmir;
end
if(model.pos_use_base_pair)
   p_mir = p_mir.*p_base_pair_mir;
   p_nonmir = p_nonmir.*p_base_pair_nonmir;
end
I = find((p_mir + p_nonmir) > 0);
p(I) = p_mir(I)./(p_mir(I)+p_nonmir(I));
[scorei,pos_ind] = max(p);
posi = pos_list(pos_ind);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function p_loopdist = loopdist_prob(pos_list, model, endbulgesi);
% calculates the probability of each position in the list based on distance from loop
```

```
%uses gaussian probability distribution
seq_size = length(endbulgesi);
lb = find(endbulgesi);
eb_begin = lb(1);
eb_end = lb(end);
zloopdist = zeros(size(pos_list));  %standardized variables
side = sign(pos_list - eb_begin);
lup = find(side == -1);
zloopdist(lup) = (eb_begin - pos_list(lup) - model.pos_upper_mean_dist)/model.pos_upper_std_dist;
llw = find(side == 1);
zloopdist(llw) = (pos_list(llw)-eb_end - model.pos_lower_mean_dist)/model.pos_lower_std_dist;
p_loopdist = exp(-0.5*zloopdist.^2);
p_loopdist = p_loopdist/sum(p_loopdist);
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%
function [p_nuc_mir,p_nuc_nonmir] = nuc_pos_prob(pos_list,model,seqsi);
p1_nuc_mir = model.pos_p1_nuc_mir;
p2_nuc_mir = model.pos_p2_nuc_mir;
p1_nuc_nonmir = model.pos_p1_nuc_nonmir;
p2_nuc_nonmir = model.pos_p2_nuc_nonmir;
win_len = model.win_len;
p_nuc_mir = zeros(size(pos_list));
p_nuc_nonmir = zeros(size(pos_list));
for i=1:length(pos_list)
  pos = pos_list(i);
  win_inds = pos:min([pos+win_len-1,length(seqsi)]);
  win_len_actual = length(win_inds);
  winseq = seqsi(win_inds);

  %multiply probabilities of single nucleotides in window 'win'
  if model.pos_nuc_order == 1
    %1 gram
    p_nuc_i = 1;
    for j = 1:win_len_actual
      p_nuc_i = p_nuc_i * p1_nuc_mir(j,winseq(j));
    end
  else
    %2 gram
    p_nuc_i = p1_nuc_mir(1,winseq(1));
    for j = 1:win_len_actual-1
      p_nuc_i = p_nuc_i * ...
        p2_nuc_mir(j,winseq(j),winseq(j+1))/p1_nuc_mir(j,winseq(j));
    end
  end
  %normalize by window length
  p_nuc_mir(i) = p_nuc_i^(win_len/win_len_actual);
  %calculate p(win given nonmir)
  if model.pos_nuc_order == 1
   p_nuc_i = 1;
```

```
  for j = 1:win_len_actual
   p_nuc_i = p_nuc_i * p1_nuc_nonmir(winseq(j));
    end
  else
    p_nuc_i = p1_nuc_nonmir(1,winseq(1));
  for j = 1:win_len_actual-1
   p_nuc_i = p_nuc_i * p2_nuc_nonmir(j,winseq(j),winseq(j+1))/p1_nuc_nonmir(j,winseq(j));
    end
  end
  %normalize by window length
  p_nuc_nonmir(i) = p_nuc_i^(win_len/win_len_actual);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%
function [p_bulge_mir,p_bulge_nonmir] = bulge_pos_prob(pos_list,model,bulges1i,bulges2i);
win_len = model.win_len;
if(model.pos_bulge == 1)
  pb_mir = model.pos_pb1_mir;
  pb_nonmir = model.pos_pb1_nonmir;
  bulges = bulges1i;
elseif(model.pos_bulge == 2)
  pb_mir = model.pos_pb2_mir;
  pb_nonmir = model.pos_pb2_nonmir;
  bulges = bulges2i;
elseif(model.pos_bulge == 0)
  pb_mir = model.pos_pbtot_mir;
  pb_nonmir = model.pos_pbtot_nonmir;
  bulges = bulges1i+bulges2i;
else
  error('model.pos_bulge must be 1 2 or 0');
end
p_bulge_mir = zeros(size(pos_list));
p_bulge_nonmir = zeros(size(pos_list));
for i=1:length(pos_list)
  pos = pos_list(i);
  win_inds = pos:min([pos+win_len-1,length(bulges)]);
  win_len_actual = length(win_inds);
  winbulges = bulges(win_inds);
  J0 = find(winbulges == 0);
  J1 = find(winbulges);
  p_bulge_i = prod(pb_mir(J1)) * prod(1-pb_mir(J0));
  p_bulge_mir(i) = p_bulge_i^(win_len/win_len_actual);
  p_bulge_i = prod(pb_nonmir(J1)) * prod(1-pb_nonmir(J0));
  p_bulge_nonmir(i) = p_bulge_i^(win_len/win_len_actual);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%
function [p_base_pair_mir,p_base_pair_nonmir] = base_pair_prob(pos_list,model,seqsi,ai);
win_len = model.win_len;
p_bp_mir = model.p_bp_mir;
```

```
p_bp_nonmir = model.p_bp_nonmir;
seqbp = nuc2bp(seqsi,ai,model.pos_base_pair_states);
p_base_pair_mir = zeros(size(pos_list));
p_base_pair_nonmir = zeros(size(pos_list));
for i=1:length(pos_list)
  pos = pos_list(i);
  win_inds = pos:min([pos+win_len-1,length(seqsi)]);
  win_len_actual = length(win_inds);
  pmir_i = 1;
  pnonmir_i = 1;
  for j = 1:model.pos_base_pair_states
    pmir_i = pmir_i * p_bp_mir(j)^sum(seqbp(win_inds) == j);
    pnonmir_i = pnonmir_i * p_bp_nonmir(j)^sum(seqbp(win_inds) == j);
  end
  p_base_pair_mir(i) =  pmir_i^(win_len/win_len_actual);
  p_base_pair_nonmir(i) =  pnonmir_i^(win_len/win_len_actual);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%
function [win_pos,win_score] = bayes_predict_win(model,seqs,anti_inds,bulges1,bulges2,endbulges)
%[win_pos,score] = bayes_predict_win(model,seqs,anti_inds,bulges1,bulges2,endbulges)
% find the best window position by its matching to the bayesian model
mfes = anti_inds_to_mfe(anti_inds);
for i = 1:length(seqs)
  %disp(num2str(i));
  [win_posi, win_scorei] = bayes_predict_win_i(model,seqs{i},mfes{i},anti_inds{i},bulges1{i},bulges2{i},endbulges{i});
  win_pos(i) = win_posi;
  win_score(i) = win_scorei;
end
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [win_pos, win_score] = bayes_predict_win_i(model,seqsi,mfei,ai,bulges1i,bulges2i, endbulgesi);
p_loopdist = loopdist_bp_prob_normal(model,mfei);
[p_num_bps_mir,p_num_bps_nonmir] = num_bps_prob_hist(model,mfei,ai);
[p_win_sym_mir,p_win_sym_nonmir] = win_sym_prob(model,mfei,ai);
[p_pos_bulge_mir,p_pos_bulge_nonmir] = win_bulges_pos_prob(model,mfei,bulges1i,bulges2i,0);
[p_base_pair_mir,p_base_pair_nonmir] = win_base_pair_prob(model,mfei,ai,seqsi);
[p_nuc_mir,p_nuc_nonmir] = win_nuc_positional_prob_sw(model,seqsi,mfei);
p_mir = ones(1,size(mfei,1));
p_nonmir = ones(1,size(mfei,1));
if(model.win_use_loopdist)
  p_mir = p_mir.*p_loopdist;
  p_nonmir = p_nonmir.*(1-p_loopdist);
end
if(model.win_use_num_bps)
  p_mir = p_mir.*p_num_bps_mir;
  p_nonmir = p_nonmir.*p_num_bps_nonmir;
end
if(model.win_use_win_sym)
```

```
    p_mir = p_mir.*p_win_sym_mir;
    p_nonmir = p_nonmir.*p_win_sym_nonmir;
  end
  if(model.win_use_pos_bulge)
    p_mir = p_mir.*p_pos_bulge_mir;
    p_nonmir = p_nonmir.*p_pos_bulge_nonmir;
  end
  if(model.win_use_base_pair)
    p_mir = p_mir.*p_base_pair_mir;
    p_nonmir = p_nonmir.*p_base_pair_nonmir;
  end
  if(model.win_use_nuc)
    p_mir = p_mir.*p_nuc_mir;
    p_nonmir = p_nonmir.*p_nuc_nonmir;
  end
  I = find((p_mir + p_nonmir) > 0);
  p(I) = p_mir(I)./(p_mir(I)+p_nonmir(I));
  [win_score,win_pos] = max(p);
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function p_loopdist = loopdist_bp_prob_normal(model,mfe);
  n_bps = size(mfe,1);
  wp = 1:n_bps;
  zloopdist = ((n_bps - wp) - model.mean_loopdist_bp)/model.std_loopdist_bp;
  zloopdist(1:model.min_win_bp-1) = 0; % illegal windows.
  p_loopdist = exp(-0.5*zloopdist.^2);
  p_loopdist = p_loopdist/sum(p_loopdist);
  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  function [p_num_bps_mir,p_num_bps_nonmir] = num_bps_prob_hist(model,mfe,ai);
  win_len = model.win_len;
  n_bps = size(mfe,1);
  p_num_bps_mir = zeros(1,n_bps);
  p_num_bps_nonmir = zeros(1,n_bps);
  is_paired = (ai~=0);
  for wp = model.min_win_bp:n_bps
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
    pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
    win5inds = pos5_on_arm5:pos3_on_arm5;
    win3inds = pos5_on_arm3:pos3_on_arm3;
    numpaired5 = sum(is_paired(win5inds));
    numpaired3 = sum(is_paired(win3inds));
    num_bps_i = min(numpaired5,numpaired3);
    % mir
    tt = find(model.win_num_bps_mir_vals == num_bps_i);
    if(tt)
      p_num_bps_mir_i = model.win_num_bps_mir_ps(tt);
    else
```

```
    p_num_bps_mir_i = 0;
  end
  p_num_bps_mir_i = p_num_bps_mir_i*(win_len/mean(length(win5inds),length(win3inds)));
  p_num_bps_mir(wp) = p_num_bps_mir_i;
  % nonmir
  tt = find(model.win_num_bps_nonmir_vals == num_bps_i);
  if(tt)
    p_num_bps_nonmir_i = model.win_num_bps_nonmir_ps(tt);
  else
    p_num_bps_nonmir_i = 0;
  end
  p_num_bps_nonmir_i = p_num_bps_nonmir_i*(win_len/mean(length(win5inds),length(win3inds)));
  p_num_bps_nonmir(wp) = p_num_bps_nonmir_i;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [p_win_sym_mir,p_win_sym_nonmir] = win_sym_prob(model,mfe,ai);
win_len = model.win_len;
n_bps = size(mfe,1);
p_win_sym_mir = zeros(1,n_bps);
p_win_sym_nonmir = zeros(1,n_bps);
is_paired = (ai~=0);
for wp = model.min_win_bp:n_bps
  pos3_on_arm5 = mfe(wp,1);
  pos5_on_arm3 = mfe(wp,2);
  pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
  pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
  win5inds = pos5_on_arm5:pos3_on_arm5;
  win3inds = pos5_on_arm3:pos3_on_arm3;
  numunpaired5 = sum(~is_paired(win5inds));
  numunpaired3 = sum(~is_paired(win3inds));
  win_sym_i = abs(numunpaired5-numunpaired3);
  % mir
  tt = find(model.win_sym_mir_vals == win_sym_i);
  if(tt)
    p_win_sym_mir_i = model.win_sym_mir_ps(tt);
  else
    p_win_sym_mir_i = 0;
  end
  p_win_sym_mir_i = p_win_sym_mir_i*sqrt(win_len/mean(length(win5inds),length(win3inds)));
  p_win_sym_mir(wp) = p_win_sym_mir_i;
  % nonmir
  tt = find(model.win_sym_nonmir_vals == win_sym_i);
  if(tt)
    p_win_sym_nonmir_i = model.win_sym_nonmir_ps(tt);
  else
    p_win_sym_nonmir_i = 0;
  end
  p_win_sym_nonmir_i = p_win_sym_nonmir_i*sqrt(win_len/mean(length(win5inds),length(win3inds)));
  p_win_sym_nonmir(wp) = p_win_sym_nonmir_i;
```

```
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [p_pos_bulge_mir,p_pos_bulge_nonmir] = win_bulges_pos_prob(model,mfe,bulges1i,bulges2i,use_avg);
bulge_flag = model.win_bulge;
win_len = model.win_len;
n_bps = size(mfe,1);
p_pos_bulge_mir = zeros(1,n_bps);
p_pos_bulge_nonmir = zeros(1,n_bps);
pb_arm5_mir = model.win_bulge_posit_arm5_mir;
pb_arm3_mir = model.win_bulge_posit_arm3_mir;
pb1_arm5_mir = model.win_bulge1_posit_arm5_mir;
pb1_arm3_mir = model.win_bulge1_posit_arm3_mir;
pb2_arm5_mir = model.win_bulge2_posit_arm5_mir;
pb2_arm3_mir = model.win_bulge2_posit_arm3_mir;
pb_arm5_nonmir = model.win_bulge_posit_arm5_nonmir;
pb_arm3_nonmir = model.win_bulge_posit_arm3_nonmir;
pb1_arm5_nonmir = model.win_bulge1_posit_arm5_nonmir;
pb1_arm3_nonmir = model.win_bulge1_posit_arm3_nonmir;
pb2_arm5_nonmir = model.win_bulge2_posit_arm5_nonmir;
pb2_arm3_nonmir = model.win_bulge2_posit_arm3_nonmir;
if(use_avg)
  pb_mir = 0.5*(pb_arm5_mir+pb_arm3_mir);
  pb_arm5_mir = pb_mir;
  pb_arm3_mir = pb_mir;
  pb1_mir = 0.5*(pb1_arm5_mir+pb1_arm3_mir);
  pb1_arm5_mir = pb1_mir;
  pb1_arm3_mir = pb1_mir;
  pb2_mir = 0.5*(pb2_arm5_mir+pb2_arm3_mir);
  pb2_arm5_mir = pb2_mir;
  pb2_arm3_mir = pb2_mir;
  pb_nonmir = 0.5*(pb_arm5_nonmir+pb_arm3_nonmir);
  pb_arm5_nonmir = pb_nonmir;
  pb_arm3_nonmir = pb_nonmir;
  pb1_nonmir = 0.5*(pb1_arm5_nonmir+pb1_arm3_nonmir);
  pb1_arm5_nonmir = pb1_nonmir;
  pb1_arm3_nonmir = pb1_nonmir;
  pb2_nonmir = 0.5*(pb2_arm5_nonmir+pb2_arm3_nonmir);
  pb2_arm5_nonmir = pb2_nonmir;
  pb2_arm3_nonmir = pb2_nonmir;
end
if(bulge_flag == 1)
  pb_arm5_mir = pb1_arm5_mir;
  pb_arm3_mir = pb1_arm3_mir;
  pb_arm5_nonmir = pb1_arm5_nonmir;
  pb_arm3_nonmir = pb1_arm3_nonmir;
  bulgesi = bulges1i;
elseif(bulge_flag == 2)
  pb_arm5_mir = pb2_arm5_mir;
  pb_arm3_mir = pb2_arm3_mir;
```

```
  pb_arm5_nonmir = pb2_arm5_nonmir;
  pb_arm3_nonmir = pb2_arm3_nonmir;
  bulgesi = bulges2i;
else
  % just use the total pb.
  bulgesi = bulges1i+bulges2i;
end
for wp = model.min_win_bp:n_bps
  pos3_on_arm5 = mfe(wp,1);
  pos5_on_arm3 = mfe(wp,2);
  pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
  pos3_on_arm3 = min(length(bulgesi),pos5_on_arm3+win_len-1);
  win5 = bulgesi(pos3_on_arm5:-1:pos5_on_arm5); % always start from loop side
  win3 = bulgesi(pos5_on_arm3:pos3_on_arm3);
  win5_len_actual = length(win5);
  win3_len_actual = length(win3);

  J0 = find(win5 == 0);
  J1 = find(win5);
  p_bulges5_mir_i = prod(pb_arm5_mir(J1)) * prod(1-pb_arm5_mir(J0));
  p_bulges5_mir_i = p_bulges5_mir_i^(win_len/win5_len_actual);
  p_bulges5_nonmir_i = prod(pb_arm5_nonmir(J1)) * prod(1-pb_arm5_nonmir(J0));
  p_bulges5_nonmir_i = p_bulges5_nonmir_i^(win_len/win5_len_actual);
  J0 = find(win3 == 0);
  J1 = find(win3);
  p_bulges3_mir_i = prod(pb_arm3_mir(J1)) * prod(1-pb_arm3_mir(J0));
  p_bulges3_mir_i = p_bulges3_mir_i^(win_len/win3_len_actual);
  p_bulges3_nonmir_i = prod(pb_arm3_nonmir(J1)) * prod(1-pb_arm3_nonmir(J0));
  p_bulges3_nonmir_i = p_bulges3_nonmir_i^(win_len/win3_len_actual);

  p_pos_bulge_mir(wp) = sqrt(p_bulges5_mir_i*p_bulges3_mir_i);
  p_pos_bulge_nonmir(wp) = sqrt(p_bulges5_nonmir_i*p_bulges3_nonmir_i);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [p_base_pair_mir,p_base_pair_nonmir] = win_base_pair_prob(model,mfe,ai,seq);
win_len = model.win_len;
base_pair_states = model.win_base_pair_states;
p_bp_arm5_mir = model.win_base_pair_arm5_mir;
p_bp_arm3_mir = model.win_base_pair_arm3_mir;
p_bp_arm5_nonmir = model.win_base_pair_arm5_nonmir;
p_bp_arm3_nonmir = model.win_base_pair_arm3_nonmir;
n_bps = size(mfe,1);
p_base_pair = zeros(1,n_bps);
t1{1} = seq;
t2{1} = ai;
t3 = nuc2bp(t1,t2,base_pair_states);
seqbp = t3{1};
for wp = model.min_win_bp:n_bps
  pos3_on_arm5 = mfe(wp,1);
```

```
  pos5_on_arm3 = mfe(wp,2);
  pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
  pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
  win5inds = (pos5_on_arm5:pos3_on_arm5);
  win3inds = (pos5_on_arm3:pos3_on_arm3);
  % mir
  p5_mir_i = 1;
  p3_mir_i = 1;
  for j = 1:base_pair_states
    p5_mir_i = p5_mir_i * p_bp_arm5_mir(j)^sum(seqbp(win5inds) == j);
    p3_mir_i = p3_mir_i * p_bp_arm3_mir(j)^sum(seqbp(win3inds) == j);
  end
  p5_mir_i = p5_mir_i.^(win_len/length(win5inds));
  p3_mir_i = p3_mir_i.^(win_len/length(win3inds));
  p_base_pair_mir(wp) =  sqrt(p5_mir_i*p3_mir_i);
  % nonmir
  p5_nonmir_i = 1;
  p3_nonmir_i = 1;
  for j = 1:base_pair_states
    p5_nonmir_i = p5_nonmir_i * p_bp_arm5_nonmir(j)^sum(seqbp(win5inds) == j);
    p3_nonmir_i = p3_nonmir_i * p_bp_arm3_nonmir(j)^sum(seqbp(win3inds) == j);
  end
  p5_nonmir_i = p5_nonmir_i.^(win_len/length(win5inds));
  p3_nonmir_i = p3_nonmir_i.^(win_len/length(win3inds));
  p_base_pair_nonmir(wp) =  sqrt(p5_nonmir_i*p3_nonmir_i);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [p_nuc_mir,p_nuc_nonmir] = win_nuc_positional_prob_sw(model,seq,mfe);
% ook at AT as one thing and at CG as one
% for now implemented only 1gram of this version
win_len = model.win_len;
win_len_common = min(win_len,model.win_nuc_pos_win);
p1_5_mir = model.win_nuc_pos_p1_5_mir;
p2_5_mir = model.win_nuc_pos_p2_5_mir;
p1_3_mir = model.win_nuc_pos_p1_3_mir;
p2_3_mir = model.win_nuc_pos_p2_3_mir;
p1_5_nonmir = model.win_nuc_pos_p1_5_nonmir;
p2_5_nonmir = model.win_nuc_pos_p2_5_nonmir;
p1_3_nonmir = model.win_nuc_pos_p1_3_nonmir;
p2_3_nonmir = model.win_nuc_pos_p2_3_nonmir;
p1_5_mir = transform_p1(p1_5_mir);
p1_3_mir = transform_p1(p1_3_mir);
p1_5_nonmir = transform_p1(p1_5_nonmir);
p1_3_nonmir = transform_p1(p1_3_nonmir);
p2_5_mir = transform_p2(p2_5_mir);
p2_3_mir = transform_p2(p2_3_mir);
p2_5_nonmir = transform_p2(p2_5_nonmir);
p2_3_nonmir = transform_p2(p2_3_nonmir);
n_bps = size(mfe,1);
```

```
for wp = model.min_win_bp:n_bps
  pos3_on_arm5 = mfe(wp,1);
  pos5_on_arm3 = mfe(wp,2);
  pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
  pos3_on_arm3 = min(length(seq),pos5_on_arm3+win_len-1);
  win5inds = (pos5_on_arm5:pos3_on_arm5);
  win3inds = (pos5_on_arm3:pos3_on_arm3);

  seq5_sw = transform_to_sw(seq(win5inds));
  seq3_sw = transform_to_sw(seq(win3inds));

  win5_len_actual = min(model.win_nuc_pos_win,length(seq5_sw));
  win3_len_actual = min(model.win_nuc_pos_win,length(seq3_sw));

  % mir
  if model.win_nuc_order == 1
    %1 gram
    p5_i = 1;
    for j = 1:win5_len_actual
      p5_i = p5_i * p1_5_mir(j,seq5_sw(j));
    end
    p3_i = 1;
    for j = 1:win3_len_actual
      p3_i = p3_i * p1_3_mir(j,seq3_sw(j));
    end
  else
    %2 gram
    p5_i = p1_5_mir(1,seq5_sw(1));
    for j = 1:win5_len_actual-1
      p5_i = p5_i * p2_5_mir(j,seq5_sw(j),seq5_sw(j+1))/p1_5_mir(j,seq5_sw(j));
    end
    p3_i = p1_3_mir(1,seq3_sw(1));
    for j = 1:win3_len_actual-1
      p3_i = p3_i * p2_3_mir(j,seq3_sw(j),seq3_sw(j+1))/p1_3_mir(j,seq3_sw(j));
    end
  end
  p5_i = p5_i.^(win_len_common/win5_len_actual);
  p3_i = p3_i.^(win_len_common/win3_len_actual);
  p_nuc_mir(wp) =  sqrt(p5_i*p3_i);

  % nonmir
  if model.win_nuc_order == 1
    %1 gram
    p5_i = 1;
    for j = 1:win5_len_actual
      p5_i = p5_i * p1_5_nonmir(j,seq5_sw(j));
    end
    p3_i = 1;
    for j = 1:win3_len_actual
      p3_i = p3_i * p1_3_nonmir(j,seq3_sw(j));
```

```
    end
  else
    %2 gram
    p5_i = p1_5_nonmir(1,seq5_sw(1));
    for j = 1:win5_len_actual-1
      p5_i = p5_i * p2_5_nonmir(j,seq5_sw(j),seq5_sw(j+1))/p1_5_nonmir(j,seq5_sw(j));
    end
    p3_i = p1_3_nonmir(1,seq3_sw(1));
    for j = 1:win3_len_actual-1
      p3_i = p3_i * p2_3_nonmir(j,seq3_sw(j),seq3_sw(j+1))/p1_3_nonmir(j,seq3_sw(j));
    end
  end
  p5_i = p5_i.^(win_len_common/win5_len_actual);
  p3_i = p3_i.^(win_len_common/win3_len_actual);
  p_nuc_nonmir(wp) =  sqrt(p5_i*p3_i);
end
function s = transform_to_sw(seq)
for i=1:length(seq)
  if(seq(i)==1 | seq(i)==3)
    s(i)=1;
  else
    s(i)=2;
  end
end
function p1 = transform_p1(p1_in)
p1_new(1,:) = mean([p1_in(:,1),p1_in(:,3)]');
p1_new(2,:) = mean([p1_in(:,2),p1_in(:,4)]');
p1 = p1_new';
function p2 = transform_p2(p2_in)
Ns = size(p2_in,2);
for j=1:size(p2_in,1)
  tt = reshape(p2_in(j,:,:),Ns,Ns);
  ttt(:,1) = (mean([tt(:,1),tt(:,3)]'))';
  ttt(:,2) = (mean([tt(:,2),tt(:,4)]'))';
  tttt(1,:) = mean([ttt(1,:);ttt(3,:)]);
  tttt(2,:) = mean([ttt(2,:);ttt(4,:)]);
  p2_new(j,:,:) = tttt;
end
p2 = p2_new;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function positions = get_possible_positions(model,mfes,endbulges,win_pos)
% function positions = get_possible_positions(model,mfes,endbulges,win_pos)
% positions{i} a list of possible positions given the window position win_pos(i)
% for each arm gives pos5 of the window on that arm plus model.possible_pos_back
% positions back and model.possible_pos_fwd positions fwd.
% will also work with win_pos of length=1 and enbulges being a vector instead of a cell
win_len = model.win_len;
naway = model.possible_pos_away;
nto = model.possible_pos_to;
```

```
if(naway<0 | nto<0)
  error('model.possible_pos_away and model.possible_pos_to must be nonnegative')
end
if(length(win_pos)==1)
  tt{1} = endbulges;
  endbulges = tt;
  ttt{1} = mfes;
  mfes = ttt;
end
for i=1:length(win_pos)
  wp = win_pos(i);
  endbulgesi = endbulges{i};
  mfe = mfes{i};
  pos3_on_arm5 = mfe(wp,1);
  pos5_on_arm3 = mfe(wp,2);
  pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);

  t5 = [max(1,pos5_on_arm5-naway) : pos5_on_arm5+nto];
  t3 = [pos5_on_arm3-nto : min(length(endbulgesi),pos5_on_arm3+naway)];
  % remove indices sitting on end bulge
  lb = find(endbulgesi);
  positions{i} = setdiff([t5,t3],lb);
end
function win_mirpos = get_win_pos_v1(mfes,anti_inds,mirpos,mirlen)
% function win_mirpos = get_win_pos(mfes,anti_inds,mirpos,mirlen)
% returns win_mirpos in index of basepair (from legs not loop).
% i.e. mfe(win_mirpos,1) is the nuc pos on the 5 arm
% for mir on arm3 returns the closest bp from its mirpos towards the legs
% for mir on arm5 returns the closest bp from its END (mirpos+mirlen-1) towards the legs
% also towards the legs
for i=1:length(mirpos)
  pos5 = mirpos(i);
  pos3 = pos5+mirlen(i)-1;
  mfe = mfes{i};
  arm5 = mfe(:,1);
  arm3 = mfe(:,2);
  eb_start = arm5(end)+1;
  eb_end = arm3(end)-1;
  eb_len = eb_end-eb_start+1;
  side5 = (pos5<eb_start);
  ai = anti_inds{i};
  is_paired = (ai~=0);
  if(side5)
    k=0;
    while(~is_paired(pos3-k))
      k=k+1;
    end
    win_mirpos(i) = find(arm5==(pos3-k));
  else
    k=0;
```

```
    while(~is_paired(pos5+k))
      k=k+1;
    end
    win_mirpos(i) = find(arm3==(pos5+k));
  end
  if(isempty(win_mirpos(i)))
    error('get_win_pos: fatal error. aborting.');
  end
end
function strseq = int2nuc(intseq, ncase)
%strseq = int2nuc(intseq, ncase)
%convert a sequence of '1 2 3 4' into  'A C T G'  or 'a c t g'
% ncase = uppercase | lowercase
if(isletter(intseq(1)))
  strseq = intseq;
  return;
end
if nargin == 1
  ncase = 'uppercase';
end
if strcmp(ncase,'uppercase')
  nucs = 'ACTG';
elseif strcmp(ncase,'lowercase')
  nucs = 'actg';
end
strseq = char(size(intseq));
for i = 1:length(intseq)
  strseq(i) = nucs(intseq(i));
end
return
function [yside, yprec2] = interpolate_prob_new(score, fitfile);
%[yside, yprec2] = interpolate_prob_new(score, fitfile);
% load the parameters for interpolation
load(fitfile);
%interpolate
yside = interp1(xs,ys,score,'linear');
yprec2 = interp1(xp2,yp2,score,'linear');
% extrapolate if necessary
if(min(xs)==xs(1)) % x is increasing
  yside(score<xs(1)) = ys(1);
  yprec2(score<xp2(1)) = yp2(1);
  yside(score>xs(end)) = ys(end);
  yprec2(score>xp2(end)) = yp2(end);
else % x is decreasing
  yside(score>xs(1)) = ys(1);
  yprec2(score>xp2(1)) = yp2(1);
  yside(score<xs(end)) = ys(end);
  yprec2(score<xp2(end)) = yp2(end);
end
returnfunction [mean_dist,std_dist] = loopdist_bp_model_normal(win_pos,mfes)
```

```
for i=1:length(win_pos)
  n_bps = size(mfes{i},1);
  loopdist(i) = n_bps - win_pos(i);
end
% cut off outliers
lp = prctile(loopdist,[2.5 97.5]);
l = find(loopdist >= lp(1) & loopdist <=lp(2));
mean_dist = mean(loopdist(l));
std_dist = std(loopdist(l));
%figure;hist(loopdist,[0:max(loopdist+1)]);title('loopdist training');function
[upper_mean_dist,upper_std_dist,lower_mean_dist,lower_std_dist] = loopdist_model(pos, endbulges)
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  side(i) = sign(pos(i) - eb(1)) ;
  loopdist(i) = 0.5* ( (1-side(i))*(eb(1) - pos(i)) + ...
    (1+side(i))*(pos(i)-eb(length(eb))));
end
%keyboard
%upper strand
l = find(side == -1);
% cut off outliers
lp = prctile(loopdist(l),[2.5 97.5]);
l = find(side == -1 & loopdist > lp(1) & loopdist <lp(2));
upper_mean_dist = mean(loopdist(l));
upper_std_dist = std(loopdist(l));
%lower strand
l = find(side == 1);
% cut off outliers
lp = prctile(loopdist(l),[2.5 97.5]);
l = find(side == 1 & loopdist > lp(1) & loopdist <lp(2));
lower_mean_dist = mean(loopdist(l));
lower_std_dist = std(loopdist(l));
return
if(~exist('maxd'))
  maxd = 4;
end
randomize=0;
filename =['C:\rosetta\data_baseline_29_7\clust_proto_' num2str(maxd) '_' set_name '.txt'];
clust_proto = load(filename);
if length(clust_proto) ~= length(palseq)
  error('clust_proto wrong size');
end
if exist('randomize')
  if randomize == 1
    error('should load training set with randomize = 0 option');
  end
end
if(~exist('param_file'))
  params_tests;
else
```

```
  eval(param_file);
end
model = model_params;
mfes = anti_inds_to_mfe(anti_inds);
% win_pos(i) is the position of the window corresponding to mir i
win_pos = get_win_pos_v1(mfes,anti_inds,mirpos,mirlen);
n_all = length(palseq);
examples = find(clust_proto==1);
length(examples)
for i=1:length(examples)
  i
  bs = examples(i);% test set
 bt = setdiff(examples, bs);% train set

  model =
bayes_learn_win(palseq(bt),anti_inds(bt),bulges1(bt),bulges2(bt),endbulges(bt),mirpos(bt),mirlen(bt),model);
  model =
bayes_learn_pos_given_win(palseq(bt),anti_inds(bt),bulges1(bt),bulges2(bt),endbulges(bt),mirpos(bt),mirlen(bt),mode
l);

  [win_pos_estm,win_scorem] =
bayes_predict_win(model,palseq(bs),anti_inds(bs),bulges1(bs),bulges2(bs),endbulges(bs));
  win_pos_est(bs) = win_pos_estm;
  win_score(bs) =win_scorem;

  % use estimated win_pos for prediction of pos!
  [pos_estm,pos_scorem]
=bayes_predict_pos_given_win(palseq(bs),win_pos_est(bs),anti_inds(bs),bulges1(bs),bulges2(bs),endbulges(bs),mo
del);
  pos_est(bs) = pos_estm;
  pos_score(bs) = pos_scorem;
end
%modelrandomize=1;
if randomize
  rand('state',randstate);
  %rand('state',sum(100*clock));
 disp('performing randomized permutation');
 l = randperm(length(palseq));
 bulges1 = bulges1(l);
  bulges2 = bulges2(l);
  anti_inds = anti_inds(l);
  endbulges = endbulges(l);
  pal_id = pal_id(l);
  energy = energy(l);
 palseq = palseq(l);
  mirseq = mirseq(l);
  mirlen = mirlen(l);
  mirpos = mirpos(l);
  mfes = mfes(l);
end
```

```
if(~exist('mfold'))
  mfold = 3;
end
eval(param_file);
model = model_params;
n_all = length(palseq);
bins = round(0:n_all/mfold:n_all);
bins_all = 1:n_all;
m = 1;
while m <= mfold
  disp(num2str(m));

  bs = [bins(m)+1: bins(m+1)];% test set
  bt = setdiff(bins_all , bs);% train set

  disp(' ');
  disp([ 'm = ' num2str(m)]);

  model =
bayes_learn_win(palseq(bt),anti_inds(bt),bulges1(bt),bulges2(bt),endbulges(bt),mirpos(bt),mirlen(bt),model);
  model =
bayes_learn_pos_given_win(palseq(bt),anti_inds(bt),bulges1(bt),bulges2(bt),endbulges(bt),mirpos(bt),mirlen(bt),mode
l);

  [win_pos_estm,win_scorem] =
bayes_predict_win(model,palseq(bs),anti_inds(bs),bulges1(bs),bulges2(bs),endbulges(bs));
  win_pos_est(bs) = win_pos_estm;
  win_score(bs) =win_scorem;

  % use estimated win_pos for prediction of pos!
  [pos_estm,pos_scorem]
=bayes_predict_pos_given_win(palseq(bs),win_pos_est(bs),anti_inds(bs),bulges1(bs),bulges2(bs),endbulges(bs),mo
del);
  pos_est(bs) = pos_estm;
  pos_score(bs) = pos_scorem;

  m = m+1;
end
%modelmaxd = 4;
randomize=0;
filename =['C:\rosetta\data_baseline_29_7\clust_proto_' num2str(maxd) '_' set_name '.txt'];
clust_proto = load(filename);
if length(clust_proto) ~= length(palseq)
  error('clust_proto wrong size');
end
if exist('randomize')
  if randomize == 1
    error('should load training set with randomize = 0 option');
  end
end
```

```
if(~exist('param_file'))
   params_tests;
else
   eval(param_file);
end
model = model_params;
n_all = length(palseq);
examples = find(clust_proto==1);
length(examples)
for i=1:length(examples)
   bs = examples(i);% test set
 bt = setdiff(examples, bs);% train set

   model =
bayes_learn_win(palseq(bt),anti_inds(bt),bulges1(bt),bulges2(bt),endbulges(bt),mirpos(bt),mirlen(bt),model);
   [win_pos_estm,win_scorem] =
bayes_predict_win(model,palseq(bs),anti_inds(bs),bulges1(bs),bulges2(bs),endbulges(bs));

   win_pos_est(bs) = win_pos_estm;
   win_score(bs) =win_scorem;
end
modelrandomize=1;
if randomize
   rand('state',sum(100*clock));
 disp('performing randomized permutation');
 l = randperm(length(palseq));
 bulges1 = bulges1(l);
   bulges2 = bulges2(l);
   anti_inds = anti_inds(l);
   endbulges = endbulges(l);
   pal_id = pal_id(l);
   energy = energy(l);
 palseq = palseq(l);
   mirseq = mirseq(l);
   mirlen = mirlen(l);
   mirpos = mirpos(l);
   mfes = mfes(l);
end
if(~exist('mfold'))
   mfold = 10;
end
if(~exist('param_file'))
   params_tests;
else
   eval(param_file);
end
model = model_params;
n_all = length(palseq);
bins = round(0:n_all/mfold:n_all);
bins_all = 1:n_all;
```

```
m = 1;
while m <= mfold
  disp(num2str(m));

  bs = [bins(m)+1: bins(m+1)];% test set
  bt = setdiff(bins_all , bs);% train set

  disp(' ');
  disp([ 'm = ' num2str(m)]);

  model =
bayes_learn_win(palseq(bt),anti_inds(bt),bulges1(bt),bulges2(bt),endbulges(bt),mirpos(bt),mirlen(bt),model);
  [win_pos_estm,win_scorem] =
bayes_predict_win(model,palseq(bs),anti_inds(bs),bulges1(bs),bulges2(bs),endbulges(bs));

  win_pos_est(bs) = win_pos_estm;
  win_score(bs) =win_scorem;

  m = m+1;
end
modelfunction seqsbp = nuc2bp(seqs,anti_inds,base_pair_basis)
%seqsbp = nuc2bp(seqs,anti_inds,base_pair_basis)
%transform to base pair representation
%for a 3 state model {AT,CG,TG} -> 1 2 3
%for a 6 state {AT,CG,TG,TA,GC,GT} -> 1 2 3 4 5 6
%also works if seqs is a vector and not a cell array, in which case returns a vector
if(~iscell(seqs))
  tt{1} = seqs;
  seqs = tt;
  tt{1} = anti_inds;
  anti_inds = tt;
  vecflag = 1;
else
  vecflag = 0;
end
map = zeros(4);
map(1,3) = 1;  %AT
map(2,4) = 2;  %CG
map(3,4) = 3;  %TG
if base_pair_basis == 3
  map = map+map';
else
  map(3,1) = 4;  %AT
  map(4,2) = 5;  %CG
  map(4,3) = 6;  %TG
end
seqsbp = cell(size(seqs));
for i = 1:length(seqs)
  seqsi = seqs{i};
  seqsbpi = zeros(size(seqsi));
```

```
  anti_indsi = anti_inds{i};

  I = find(anti_indsi ~= 0);
  for j = 1:length(I)
    ij = I(j);
    seqsbpi(ij) = map(seqsi(ij),seqsi(anti_indsi(ij)));
  end
  seqsbp{i} = seqsbpi;
end
if(vecflag)
  tt=seqsbp{1};
  seqsbp = tt;
end
return
function [intseq, fault_seq] = nuc2int4_new(strseq);
%[intseq, fault_seq] = nuc2int4_new(strseq)
%convert a sequence of 'A C T G' into a array of 1 2 3 4
intseq = zeros(size(strseq));
fault_seq = 0;
for i = 1:length(strseq)
  switch upper(strseq(i))
    case 'A' , intseq(i) = 1;
    case 'C' , intseq(i) = 2;
    case 'T' , intseq(i) = 3;
    case 'G' , intseq(i) = 4;
    otherwise , intseq = []; fault_seq = 1; break;
  end
end
function [p1,p2]= nucleotide_pos_model_list(model,seqs,positions);
% function [p1,p2]= nucleotide_pos_model_list(model,seqs,positions);
% learns a nucleotide positional model of a list of positions
% positions{i} is the list of positions on seqs{i}
% will work also if positions is a vector and not a cell
win_len = model.win_len;
numseqs = length(positions);
if(numseqs~=length(seqs))
  error('number of seqs differs from length(positions)');
end
% transform positions into cell if it is not so.
if(~iscell(positions))
  for i=1:numseqs
    tt{i} = positions(i);
  end
  positions = tt;
end
beta = 0.5;
Ns = 4;   %number of states
c1 = zeros(win_len,Ns);
c2 = beta*ones(win_len-1,Ns,Ns);
p1 = c1;
```

```
p2 = c2;
for i = 1:numseqs
  seq = seqs{i};
  pos_list = positions{i};
  for k = 1:length(pos_list)
    posk = pos_list(k);  %current windows anchor
    %1 gram
    for j = posk:min([posk+win_len-1 length(seq)])
      jind = j-posk+1;
      c1(jind,seq(j)) = c1(jind,seq(j)) + 1;
    end
    %2 gram
    for j = posk:min([posk+win_len-1 length(seq)])-1
      jind = j-posk+1;
      c2(jind,seq(j), seq(j+1)) = c2(jind,seq(j), seq(j+1)) + 1;
    end
  end
end
for j = 1:win_len
  p1(j,:) = c1(j,:)/sum(c1(j,:));
end
for j = 1:win_len-1
  p2(j,:) = c2(j,:)/sum(c2(j,:));
end
function [num_bps_vals,num_bps_ps] = num_bps_model_hist_list(mfes,anti_inds,model,wps)
numseqs = length(wps);
if(numseqs~=length(mfes) | numseqs~=length(anti_inds))
  error('number of seqs differs from length(wps)');
end
% transform wps into cell if it is not so.
if(~iscell(wps))
  for i=1:numseqs
    tt{i} = wps(i);
  end
  wps = tt;
end
beta = 0.5;
win_len = model.win_len;
num_bps = [];
for i=1:numseqs
  wp_list = wps{i};
  mfe = mfes{i};
  ai = anti_inds{i};
  is_paired = (ai~=0);
  for k=1:length(wp_list)
    wp = wp_list(k);
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
    pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
```

```
    numpaired5 = sum(is_paired(pos5_on_arm5:pos3_on_arm5));
    numpaired3 = sum(is_paired(pos5_on_arm3:pos3_on_arm3));
    num_bps = [num_bps,min(numpaired5,numpaired3)];
  end
end
num_bps_vals = 0:model.win_num_bins_num_bps-1;
n = hist(num_bps,num_bps_vals);
n = n+beta;
num_bps_ps = n/sum(n);
%figure;bar(num_bps_vals,num_bps_ps);title('numbps hist training');
% general params
model_params.win_len = 22; % in nts.
% win params
model_params.win_base_pair_states = 6; % this param is used only for win prediction.
model_params.min_win_bp = 14; % do not allow window to start in bp lower than this.
model_params.win_bulge = 0; % for win prediction. which bulges to look at. 1/2 - bulges1/2, else total
model_params.win_nuc_order = 2; % for positional nuc in win
model_params.win_nuc_pos_win = 15; % for nuc_positional how far in window to look. put win_len for all window.
model_params.win_num_bins_sym = model_params.win_len;
model_params.win_num_bins_num_bps = model_params.win_len;
model_params.win_use_loopdist = 1;
model_params.win_use_win_sym = 1;
model_params.win_use_pos_bulge = 1;
model_params.win_use_num_bps = 1;
model_params.win_use_base_pair = 1;
model_params.win_use_nuc = 1;
% for prediction of pos_given_win
% if the below 2 params are both 0 only looks at the pos5.
model_params.possible_pos_away = 0; % how many to go from 5pos in direction away from loop
        % when searching for positions.
        % note that 0 doesn't go back at all.model_params.
model_params.possible_pos_to = 0; % same but towards loop
model_params.pos_nuc_order = 2; % nuc order for positional nuc
model_params.win_len_for_pos_nuc = 3; % size of win to count nucs. if win_len then looks at whole window
model_params.pos_bulge = 0; % which bulges to look at 1,2 or 0 for the total.
model_params.pos_base_pair_states = 6;
model_params.pos_use_loopdist = 1;
model_params.pos_use_pos_nuc = 1;
model_params.pos_use_pos_bulge = 0;
model_params.pos_use_base_pair = 1;

% general params
model_params.win_len = 22; % in nts.
% win params
model_params.win_base_pair_states = 6; % this param is used only for win prediction.
model_params.min_win_bp = 14; % do not allow window to start in bp lower than this.
model_params.win_bulge = 0; % for win prediction. which bulges to look at. 1/2 - bulges1/2, else total
model_params.win_nuc_order = 2; % for positional nuc in win
model_params.win_nuc_pos_win = 15; % for nuc_positional how far in window to look. put win_len for all window.
model_params.win_num_bins_sym = model_params.win_len;
```

```
model_params.win_num_bins_num_bps = model_params.win_len;
model_params.win_use_loopdist = 1;
model_params.win_use_win_sym = 1;
model_params.win_use_pos_bulge = 1;
model_params.win_use_num_bps = 1;
model_params.win_use_base_pair = 1;
model_params.win_use_nuc = 1;
% for prediction of pos_given_win
% if the below 2 params are both 0 only looks at the pos5.
model_params.possible_pos_away = 0; % how many to go from 5pos in direction away from loop
        % when searching for positions.
        % note that 0 doesn't go back at all.model_params.
model_params.possible_pos_to = 0; % same but towards loop
model_params.pos_nuc_order = 2; % nuc order for positional nuc
model_params.win_len_for_pos_nuc = 3; % size of win to count nucs. if win_len then looks at whole window
model_params.pos_bulge = 0; % which bulges to look at 1,2 or 0 for the total.
model_params.pos_base_pair_states = 6;
model_params.pos_use_loopdist = 1;
model_params.pos_use_pos_nuc = 1;
model_params.pos_use_pos_bulge = 0;
model_params.pos_use_base_pair = 1;

function p_bp = pos_base_pair_model_list(model,seqs,anti_inds,positions)
%function p_bp = base_pair_model_list(model,seqs,anti_inds,positions)
%learns a nonpositional model of base pairs
% positions{i} is the list of positions on seqs{i}
% will work also if positions is a vector and not a cell
win_len = model.win_len;
numseqs = length(positions);
if(numseqs~=length(seqs) | numseqs~=length(anti_inds))
  error('number of seqs or anti_inds differs from length(positions)');
end
% transform positions into cell if it is not so.
if(~iscell(positions))
  for i=1:numseqs
    tt{i} = positions(i);
  end
  positions = tt;
end
seqsbp = nuc2bp(seqs,anti_inds,model.pos_base_pair_states);
c_bp = zeros(1,model.pos_base_pair_states);
for i = 1:numseqs
  seqbp = seqsbp{i};
  pos_list = positions{i};
  for k = 1:length(pos_list)
    posk = pos_list(k);  %current windows anchor
    inds = posk:min([posk+win_len-1 length(seqbp)]);
    for j = 1:model.pos_base_pair_states
      c_bp(j) = c_bp(j)+sum(seqbp(inds) == j);
    end
```

```
  end
end
p_bp = c_bp/sum(c_bp);
function [pb1,pb2,pbtot] = pos_bulge_pos_model_list(model,bulges1,bulges2,positions);
% function [pb1,pb2,pbtot] = pos_bulge_pos_model_list(model,bulges1,bulges2,positions);
% learns a bulge positional model of a list of positions
% positions{i} is the list of positions on seqs{i}
% will work also if positions is a vector and not a cell
win_len = model.win_len;
numseqs = length(positions);
if(numseqs~=length(bulges1) | numseqs~=length(bulges2))
  error('number of bulges differs from length(positions)');
end
% transform positions into cell if it is not so.
if(~iscell(positions))
  for i=1:numseqs
    tt{i} = positions(i);
  end
  positions = tt;
end
for i = 1:numseqs
  b1 = bulges1{i};
  b2 = bulges2{i};
  btot{i} = b1+b2;
end
pb1 = bulge_positional(model,bulges1,positions);
pb2 = bulge_positional(model,bulges2,positions);
pbtot = bulge_positional(model,btot,positions);
function p = bulge_positional(model,bulges,positions)
win_len = model.win_len;
c = zeros(win_len,2);
p = zeros(win_len,1);
for i = 1:length(bulges)
  bulgesi = bulges{i};
  pos_list = positions{i};
  for k = 1:length(pos_list)
    posk = pos_list(k);  %current windows anchor
    inds = posk:min([posk+win_len-1 length(bulgesi)]);
    for j=1:length(inds)
      this_ind = inds(j);
      c(j,1) = c(j,1) + bulgesi(this_ind);
      c(j,2) = c(j,2) + (1-bulgesi(this_ind));
    end
  end
end
for j = 1:win_len
   p(j) = c(j,1)/sum(c(j,:));
end
function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid(fid,seqtot,minbp)
```

```
% function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid(fid,seqtot,minbp)
% same as read_structure_withanti_fid but reads file that have before the 4 line zuker draw
% a line giving the pal_id and a line giving the energy.
% all_pal_ids is all ids read from file, whether faulty or not
% new feature: checks that draw is not messed up and if it is gives faulty seq.
% minbp is the minimal number of basepair required for a legal pal.
Mxplen = 250;  % maximal length of palindrom
counter = 0;
seq_no = 0;
seqs = cell(0);
bulges_nonsym= cell(0);
bulges_sym= cell(0);
endbulges = cell(0);
pal_id = zeros(0);
energy = zeros(0);
while ~feof(fid) & seq_no < seqtot
  this_pal_id = str2double(fgetl(fid));
  this_energy = str2double(fgetl(fid));
  structure = char(4,250);
  i = 0;
  line = fgetl(fid);
  if(isempty(line))
    line = 'emptyline';
    fault_seq_emptyline = 1;
  else
    fault_seq_emptyline = 0;
  end
  while(line(1)~='|')  % if emptyline this is always true so will go into loop
    i = i+1;
    structure(i,1:length(line)) = line;
    line = fgetl(fid);
    if(isempty(line))
      line = 'emptyline';
      fault_seq_emptyline = 1;
    end
  end
  if(i~=4)
    fault_seq_numlines = 1;
  else
    fault_seq_numlines = 0;
  end

  fault_seq_struct = 1; % guilty until proven innocent
  fault_seq_nuc = 1;
  fault_seq_minbp = 1;
  if(fault_seq_numlines == 0 & fault_seq_emptyline==0)
    [seqi, anti_indi, bulge1i, bulge2i, endbulgei,fault_seq_struct] = get_features(structure);
    if(fault_seq_struct==0)
      % this is the old bulge1 and bulge2, now need to correct that
```

```
      bulge_nonsymi=bulge1i;
      bulge_symi=bulge2i;
      for j = 1:length(seqi)
        if(bulge_nonsymi(j))
          if(bulge_symi(max(1,j-1))) % a neighbor has a bulgesym flag on
            bulge_symi(j) = 1;
            bulge_nonsymi(j) = 0;
          end
        end
      end
      for j = length(seqi):-1:1
        if(bulge_nonsymi(j))
          if(bulge_symi(min(j+1,length(seqi)))) % a neighbor has a bulgesym flag on
            bulge_symi(j) = 1;
            bulge_nonsymi(j) = 0;
          end
        end
      end
      [intseq, fault_seq_nuc] = nuc2int4_new(seqi);
      this_mfe = anti_inds_to_mfe(anti_indi);
      n_bps = size(this_mfe,1);
      if(n_bps < minbp)
        fault_seq_minbp = 1;
      else
        fault_seq_minbp = 0;
      end
    end
  end

  if (fault_seq_struct == 0 & fault_seq_nuc == 0 & fault_seq_numlines == 0 & ...
      fault_seq_emptyline == 0 & fault_seq_minbp == 0)
    seq_no = seq_no + 1;
    seqs{seq_no} = intseq;
    anti_inds{seq_no} = anti_indi;
    bulges_nonsym{seq_no} = bulge_nonsymi;
    bulges_sym{seq_no} = bulge_symi;
    endbulges{seq_no} = endbulgei;
    pal_id(seq_no) = this_pal_id;
    energy(seq_no) = this_energy;
    counter = counter + 1;
    all_pal_ids(counter) = this_pal_id;
  else
    disp(['faulty seq on pal id  ' num2str(this_pal_id)])
    if(fault_seq_emptyline)
      disp(['reason is that there was an empty line in zuker']);
    elseif(fault_seq_numlines)
      disp(['reason is that there were not 4 lines in the draw']);
    elseif(fault_seq_struct)
      disp(['reason is that draw was messed has nuc in pair and bulge at the same time']);
    elseif(fault_seq_nuc)
```

```
        disp(['reason is that there was an illegal letter in the seq']);
      elseif(fault_seq_minbp)
        disp(['reason is that there were less basepairs then minbp']);
      end
      counter = counter + 1;
      all_pal_ids(counter) = this_pal_id;
   end
end
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [seq, anti_ind, bulge1, bulge2, endbulge, fault_seq] = get_features(structure)
% get sequence as well as bulge structure
fault_seq = 0;
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
tmpmat = zeros(2,max_col);
count = 0;
for col =1: max_col
  fl = find(isletter(uphalf(:,col)));
  if (length(fl)>1);
    fault_seq = 1;
    seq=nan;anti_ind=nan;bulge1=nan;bulge2=nan;endbulge=nan;
    return;
  end;
  if ~isempty(fl)
    count = count + 1;
    seq(count) = uphalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(1,col) = 0;
    else
      tmpmat(1,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
  end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
```

```
lwhalf = structure(3:4,:);
pos = length(bulge1);
while bulge1(pos) == 1
  endbulge(pos) = 1;
  bulge1(pos) = 0;
  pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(2,col) = 0;
    else
      tmpmat(2,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
anti_ind = zeros(size(bulge1));
for col=1:max_col
  if(tmpmat(1,col))
    anti_ind(tmpmat(1,col)) = tmpmat(2,col);
    anti_ind(tmpmat(2,col)) = tmpmat(1,col);
  end
end
return
function run_2stage_2pred()
%infile = 'c:\rosetta\data_baseline_29_7\zuker_draw_h152_pipe.txt';
infile = 'C:\rosetta\criteria_for_paper\tests\Zuker_Draw_7pals.txt';
outfile = 'C:\rosetta\criteria_for_paper\tests\out_7pals.txt';
model_filename = 'model_hmdc440_sanger_09_09_03_params1.mat';
fit_filename_both = 'fitfile_mfold3_use_bothsides_hmdc440_sanger_09_09_03_params1.mat';
fit_filename_best = 'fitfile_mfold3_use_bestside_hmdc440_sanger_09_09_03_params1.mat';
fidin = fopen(infile,'r');
fidout = fopen(outfile,'w');
```

```
seqstot = 1000; %number of sequences to classify each loop
load(model_filename);
while ~feof(fidin)
  disp('reading structure...');
  [palseq,anti_inds,bulges1,bulges2,endbulges,pal_id,energy,all_pal_ids] = ...
    read_struct_minbp(fidin,seqstot,model.min_win_bp);
  mfes = anti_inds_to_mfe(anti_inds);
  [win_pos_est,win_score] = bayes_predict_win(model,palseq,anti_inds,bulges1,bulges2,endbulges);
  score = win_score;
  % use estimated win_pos for prediction of pos!
  [pos_est,pos_score]
=bayes_predict_pos_given_win(palseq,win_pos_est,anti_inds,bulges1,bulges2,endbulges,model);

  clear pos_est_arm5 pos_est_arm3 pos_est_first pos_est_second res
  for i=1:length(win_score)
    mfe = mfes{i};
    pos_est_arm5(i) = max(1,(mfe(win_pos_est(i),1) - model.win_len + 1));
    pos_est_arm3(i) = mfe(win_pos_est(i),2);
    if(pos_est(i)==pos_est_arm5(i))
      pos_est_first(i) = pos_est_arm5(i);
      pos_est_second(i) = pos_est_arm3(i);
    elseif(pos_est(i)==pos_est_arm3(i))
      pos_est_first(i) = pos_est_arm3(i);
      pos_est_second(i) = pos_est_arm5(i);
    else
      disp('something is wrong: pos_est must be either pos_est_arm5 or pos_est_arm3. giving nan!');
      pos_est_first(i) = nan;
      pos_est_second(i) = nan;
    end
  end

  % infer probabilities
  [yside, yprec2_both] = interpolate_prob_new(score, fit_filename_both);
  [yside, yprec2_best] = interpolate_prob_new(score, fit_filename_best);

  %write to file
  %seq_id0 is added so as to sequential order of sequence numbers
  res = [pal_id; pos_est_first; pos_est_second; score; yprec2_both;yprec2_best];
  fprintf(fidout, '%d %d %d %g %g %g\r\n', res);
end
fclose(fidin);
fclose(fidout);
function [pal_id,pos_est_first,pos_est_second,score,yprec2_both,yprec2_best] =
run_2stage_2pred_giveout(palseq,anti_inds,bulges1,bulges2,endbulges,pal_id,energy)
model_filename = 'model_hmdc440_sanger_09_09_03_params1.mat';
fit_filename_both = 'fitfile_mfold3_use_bothsides_hmdc440_sanger_09_09_03_params1.mat';
fit_filename_best = 'fitfile_mfold3_use_bestside_hmdc440_sanger_09_09_03_params1.mat';
load(model_filename);
mfes = anti_inds_to_mfe(anti_inds);
[win_pos_est,win_score] = bayes_predict_win(model,palseq,anti_inds,bulges1,bulges2,endbulges);
```

```
score = win_score;
% use estimated win_pos for prediction of pos!
[pos_est,pos_score]
=bayes_predict_pos_given_win(palseq,win_pos_est,anti_inds,bulges1,bulges2,endbulges,model);
for i=1:length(win_score)
  mfe = mfes{i};
  pos_est_arm5(i) = max(1,(mfe(win_pos_est(i),1) - model.win_len + 1));
  pos_est_arm3(i) = mfe(win_pos_est(i),2);
  if(pos_est(i)==pos_est_arm5(i))
    pos_est_first(i) = pos_est_arm5(i);
    pos_est_second(i) = pos_est_arm3(i);
  elseif(pos_est(i)==pos_est_arm3(i))
    pos_est_first(i) = pos_est_arm3(i);
    pos_est_second(i) = pos_est_arm5(i);
  else
    disp('something is wrong: pos_est must be either pos_est_arm5 or pos_est_arm3. giving nan!');
    pos_est_first(i) = nan;
    pos_est_second(i) = nan;
  end
end
% infer probabilities
[yside, yprec2_both] = interpolate_prob_new(score, fit_filename_both);
[yside, yprec2_best] = interpolate_prob_new(score, fit_filename_best);
data_dir = 'data_baseline_29_7';
%set_name = 'h152';
%fid = fopen(['c:\rosetta\data_baseline_29_7\zuker_draw_' set_name '_pipe.txt'],'r');
set_name = 'hmdc440_sanger_09_09_03';
fid = fopen(['c:\rosetta\data_baseline_29_7\zuker_draw_' set_name '.txt'],'r');
[palseq,anti_inds,bulges1,bulges2,endbulges,pal_id,energy,all_pal_ids] = ...
  read_structure_with_id_fid(fid,1000);
fclose(fid);
if(length(pal_id)~=length(all_pal_ids))
  error('in human data do not allow faulty seqs, take out of there');
end
mfes = anti_inds_to_mfe(anti_inds);
fname = ['c:\rosetta\data_baseline_29_7\mirseq_' set_name '.txt'];
[mirseq,mirlen] = read_seq_with_id(fname);
mirpos = locate_dicer(mirseq,palseq);
extension = [set_name '_mfold3_params1'];
param_file='params1';
params1;
model = model_params;
model = bayes_learn_win(palseq,anti_inds,bulges1,bulges2,endbulges,mirpos,mirlen,model);
model = bayes_learn_pos_given_win(palseq,anti_inds,bulges1,bulges2,endbulges,mirpos,mirlen,model);
eval(['save model_' extension '.mat model']);
mfold = 3;
mfold_cv_random;
% chooses the correct side to only test win prediction and not side prediction
for i=1:length(mirpos)
  mfe = mfes{i};
```

```
  pos_est_arm5(i) = max(1,(mfe(win_pos_est(i),1) - model.win_len + 1));
  pos_est_arm3(i) = mfe(win_pos_est(i),2);
  d5 = abs(pos_est_arm5(i)-mirpos(i));
  d3 = abs(pos_est_arm3(i)-mirpos(i));
  pos_error(i) = min(d5,d3);
  if(d3<d5)
    pos_est_side_known(i) = pos_est_arm3(i);
  else
    pos_est_side_known(i) = pos_est_arm5(i);
  end
end
score=win_score;
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est,score,mirpos,endbulges);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] = analyse_errors_bins2(pos_est,score,mirpos,endbulges,num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
eval(['print -djpeg use_bestside_' extension '.jpeg']);
eval(['save fitfile_use_bestside_' extension '.mat xs ys xp2 yp2']);
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est_side_known,score,mirpos,endbulges);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] = analyse_errors_bins2(pos_est_side_known,score,mirpos,endbulges,num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
eval(['print -djpeg use_bothsides_' extension '.jpeg']);
eval(['save fitfile_use_bothsides_' extension '.mat xs ys xp2 yp2']);
figure;
fid = fopen(['info_and_criteria_' extension '.txt'],'w');
thresh_vec = [0:0.01:1];
clf;[thresh,acc2_bestside,captures] = analyse_errors_thresh_B(pos_est,score,mirpos,endbulges,thresh_vec);
clf;[thresh,acc2_bothsides,captures] =
analyse_errors_thresh_B(pos_est_side_known,score,mirpos,endbulges,thresh_vec);
grid
legend('off')
fprintf(fid,'%%thresh\tacc2_bothsides\tacc2_bestside\tcaptures\r\n');
for i=1:length(thresh)
```

```
  fprintf(fid,'%1.4f\t%1.4f\t%1.4f\t%d\r\n',thresh(i),acc2_bothsides(i),acc2_bestside(i),captures(i));
end
fclose(fid);
data_dir = 'data_baseline_29_7';
set_name = 'h152';
fid = fopen(['c:\rosetta\data_baseline_29_7\zuker_draw_' set_name '_pipe.txt'],'r');
%set_name = 'hmdc440_sanger_09_09_03';
%fid = fopen(['c:\rosetta\data_baseline_29_7\zuker_draw_' set_name '.txt'],'r');
[palseq,anti_inds,bulges1,bulges2,endbulges,pal_id,energy,all_pal_ids] = ...
  read_structure_with_id_fid(fid,1000);
fclose(fid);
if(length(pal_id)~=length(all_pal_ids))
  error('in human data do not allow faulty seqs, take out of there');
end
mfes = anti_inds_to_mfe(anti_inds);
fname = ['c:\rosetta\data_baseline_29_7\mirseq_' set_name '.txt'];
[mirseq,mirlen] = read_seq_with_id(fname);
mirpos = locate_dicer(mirseq,palseq);
extension = [set_name '_proto4_params1'];
params1;
model = model_params;
model = bayes_learn_win(palseq,anti_inds,bulges1,bulges2,endbulges,mirpos,mirlen,model);
model = bayes_learn_pos_given_win(palseq,anti_inds,bulges1,bulges2,endbulges,mirpos,mirlen,model);
eval(['save model_' extension '.mat model']);
mfold = 3;
maxd=4;
mfold_cv_proto;
mfes_e = mfes(examples);
mirpos_e = mirpos(examples);
win_score_e = win_score(examples);
pos_est_e = pos_est(examples);
win_pos_est_e = win_pos_est(examples);
endbulges_e = endbulges(examples);
% chooses the correct side to only test win prediction and not side prediction
for i=1:length(examples)
  mfe = mfes_e{i};
  pos_est_arm5(i) = max(1,(mfe(win_pos_est_e(i),1) - model.win_len + 1));
  pos_est_arm3(i) = mfe(win_pos_est_e(i),2);
  d5 = abs(pos_est_arm5(i)-mirpos_e(i));
  d3 = abs(pos_est_arm3(i)-mirpos_e(i));
  pos_error(i) = min(d5,d3);
  if(d3<d5)
    pos_est_side_known_e(i) = pos_est_arm3(i);
  else
    pos_est_side_known_e(i) = pos_est_arm5(i);
  end
end
score_e=win_score_e;
figure
subplot(2,1,1)
```

```
res = analyse_errors_perc(pos_est_e,score_e,mirpos_e,endbulges_e);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] = analyse_errors_bins2(pos_est_e,score_e,mirpos_e,endbulges_e,num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
eval(['print -djpeg use_bestside_' extension '.jpeg']);
eval(['save fitfile_use_bestside_' extension '.mat xs ys xp2 yp2']);
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est_side_known_e,score_e,mirpos_e,endbulges_e);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] = analyse_errors_bins2(pos_est_side_known_e,score_e,mirpos_e,endbulges_e,num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
eval(['print -djpeg use_bothsides_' extension '.jpeg']);
eval(['save fitfile_use_bothsides_' extension '.mat xs ys xp2 yp2']);
figure;
fid = fopen(['info_and_criteria_' extension '.txt'],'w');
thresh_vec = [0:0.01:1];
clf;[thresh,acc2_bestside,captures] =
analyse_errors_thresh_B(pos_est_e,score_e,mirpos_e,endbulges_e,thresh_vec);
clf;[thresh,acc2_bothsides,captures] =
analyse_errors_thresh_B(pos_est_side_known_e,score_e,mirpos_e,endbulges_e,thresh_vec);
grid
legend('off')
fprintf(fid,'%%thresh\tacc2_bothsides\tacc2_bestside\tcaptures\r\n');
for i=1:length(thresh)
  fprintf(fid,'%1.4f\t%1.4f\t%1.4f\t%d\r\n',thresh(i),acc2_bothsides(i),acc2_bestside(i),captures(i));
end
fclose(fid);
function [p_bp_arm5,p_bp_arm3] = win_base_pair_model_list(mfes,anti_inds,seqs,model,wps)
numseqs = length(wps);
if(numseqs~=length(mfes) | numseqs~=length(anti_inds) | numseqs~=length(seqs))
  error('number of seqs differs from length(wps)');
end
% transform wps into cell if it is not so.
if(~iscell(wps))
  for i=1:numseqs
    tt{i} = wps(i);
  end
```

```
  wps = tt;
end
win_len = model.win_len;
base_pair_states = model.win_base_pair_states;
c_bp_arm5 = zeros(1,base_pair_states);
c_bp_arm3 = zeros(1,base_pair_states);
seqsbp = nuc2bp(seqs,anti_inds,base_pair_states);
for i = 1:numseqs
  wp_list = wps{i};
  mfe = mfes{i};
  ai = anti_inds{i};
  is_paired = (ai~=0);
  for k=1:length(wp_list)
    wp = wp_list(k);
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
    pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
    for j = 1:base_pair_states
      c_bp_arm5(j) = c_bp_arm5(j)+sum(seqsbp{i}(pos5_on_arm5:pos3_on_arm5) == j);
      c_bp_arm3(j) = c_bp_arm3(j)+sum(seqsbp{i}(pos5_on_arm3:pos3_on_arm3) == j);
    end
  end
end
p_bp_arm5 = c_bp_arm5/sum(c_bp_arm5);
p_bp_arm3 = c_bp_arm3/sum(c_bp_arm3);
function [pb_arm5,pb_arm3,pb1_arm5,pb1_arm3,pb2_arm5,pb2_arm3] = ...
  win_bulge_pos_model_list(mfes,bulges1,bulges2,model,wps)
% on both sides of window from loop end of window
% pb1 - for bulges1 pb2 - for bulges2 pb - for total
win_len = model.win_len;
numseqs = length(wps);
if(numseqs~=length(mfes) | numseqs~=length(bulges1) | numseqs~=length(bulges2))
  error('number of seqs differs from length(wps)');
end
% transform wps into cell if it is not so.
if(~iscell(wps))
  for i=1:numseqs
    tt{i} = wps(i);
  end
  wps = tt;
end
for i=1:numseqs
  wp_list = wps{i};
  mfe = mfes{i};
  bulges{i} = bulges1{i}+bulges2{i};
  inds5_i = cell(0);
  inds3_i = cell(0);
  for k=1:length(wp_list)
    wp = wp_list(k);
```

```
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
    pos3_on_arm3 = min(length(bulges{i}),pos5_on_arm3+win_len-1);
    inds5_i{k} = pos3_on_arm5:-1:pos5_on_arm5; % always start from loop side
    inds3_i{k} = pos5_on_arm3:pos3_on_arm3;
  end
  inds5{i} = inds5_i;
  inds3{i} = inds3_i;
end
pb_arm5 = bulge_positional_list(model,bulges,inds5);
pb_arm3 = bulge_positional_list(model,bulges,inds3);
pb1_arm5 = bulge_positional_list(model,bulges1,inds5);
pb1_arm3 = bulge_positional_list(model,bulges1,inds3);
pb2_arm5 = bulge_positional_list(model,bulges2,inds5);
pb2_arm3 = bulge_positional_list(model,bulges2,inds3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function p = bulge_positional_list(model,bulges,inds)
win_len = model.win_len;
c = zeros(win_len,2);
p = zeros(win_len,1);
for i = 1:length(bulges)
  bulgesi = bulges{i};
  for k = 1:length(inds{i})
    this_inds = inds{i}{k};
    for j=1:length(this_inds)
      this_ind = this_inds(j);
      c(j,1) = c(j,1) + bulgesi(this_ind);
      c(j,2) = c(j,2) + (1-bulgesi(this_ind));
    end
  end
end
for j = 1:win_len
  p(j) = c(j,1)/sum(c(j,:));
end
function [p1_5,p2_5,p1_3,p2_3] = win_nuc_positional_model_list(seqs,mfes,model,wps)
win_len = model.win_len;
numseqs = length(wps);
if(numseqs~=length(mfes) | numseqs~=length(seqs))
  error('number of seqs differs from length(wps)');
end
% transform wps into cell if it is not so.
if(~iscell(wps))
  for i=1:numseqs
    tt{i} = wps(i);
  end
  wps = tt;
end
beta = 0.5;
```

```
Ns = 4;   %number of states
c1_5 = zeros(win_len,Ns);
c2_5 = beta*ones(win_len-1,Ns,Ns);
c1_3 = zeros(win_len,Ns);
c2_3 = beta*ones(win_len-1,Ns,Ns);
for i = 1:numseqs
  wp_list = wps{i};
  mfe = mfes{i};
  seqsi = seqs{i};
  for k=1:length(wp_list)
    wp = wp_list(k);
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
    pos3_on_arm3 = min(length(seqsi),pos5_on_arm3+win_len-1);
    inds5 = pos5_on_arm5:pos3_on_arm5;
    inds3 = pos5_on_arm3:pos3_on_arm3;
    seq5 = seqsi(inds5);
    seq3 = seqsi(inds3);

    %1 gram
    for j = 1:length(seq5)
      c1_5(j,seq5(j)) = c1_5(j,seq5(j)) + 1;
    end
    %2 gram
    for j = 1:length(seq5)-1
      c2_5(j,seq5(j),seq5(j+1)) = c2_5(j,seq5(j), seq5(j+1)) + 1;
    end

    %1 gram
    for j = 1:length(seq3)
      c1_3(j,seq3(j)) = c1_3(j,seq3(j)) + 1;
    end
    %2 gram
    for j = 1:length(seq3)-1
      c2_3(j,seq3(j),seq3(j+1)) = c2_3(j,seq3(j), seq3(j+1)) + 1;
    end
  end
end
for j = 1:win_len
  p1_5(j,:) = c1_5(j,:)/sum(c1_5(j,:));
  p1_3(j,:) = c1_3(j,:)/sum(c1_3(j,:));
end
for j = 1:win_len-1
  p2_5(j,:,:) = c2_5(j,:,:)/sum(c2_3(j,:));
  p2_3(j,:,:) = c2_3(j,:,:)/sum(c2_5(j,:));
end
function  [win_sym_vals,win_sym_ps] = win_sym_model_list(mfes,anti_inds,model,wps)
numseqs = length(wps);
if(numseqs~=length(mfes) | numseqs~=length(anti_inds))
```

```
  error('number of seqs differs from length(wps)');
end
% transform wps into cell if it is not so.
if(~iscell(wps))
  for i=1:numseqs
    tt{i} = wps(i);
  end
  wps = tt;
end
beta = 0.5;
win_len = model.win_len;
win_sym = [];
for i=1:numseqs
  wp_list = wps{i};
  mfe = mfes{i};
  ai = anti_inds{i};
  is_paired = (ai~=0);
  for k=1:length(wp_list)
    wp = wp_list(k);
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
    pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
    numunpaired5 = sum(~is_paired(pos5_on_arm5:pos3_on_arm5));
    numunpaired3 = sum(~is_paired(pos5_on_arm3:pos3_on_arm3));
    win_sym = [win_sym,abs(numunpaired5-numunpaired3)];
  end
end
win_sym_vals = 0:model.win_num_bins_sym-1;
n = hist(win_sym,win_sym_vals);
n = n+beta;
win_sym_ps = n/sum(n);
%figure;bar(win_sym_vals,win_sym_ps);title('win sym training');
```

```
function res = analyse_errors_perc(pos_estimated,score,pos, endbulges)
%analyse_errors_perc(pos_estimated,score,pos, endbulges)
% measure the distribution of erros
N = 100;
perc = [1:-1/N:0]*100;
thresh = prctile(score, perc);
accuracy = zeros(0);
correct_side_dist1 = zeros(0); %correct size, distance = 1;
correct_side_dist2 = zeros(0);
correct_side_disth = zeros(0);
wrong_side = zeros(0);
fraction = zeros(0);
count = 0;
N = length(pos);
for i = 1:length(endbulges)
  eb = find(endbulges{i});
  correct_side(i) = 0.5*( 1 + sign((pos_estimated(i) - eb(1))*(pos(i) -eb(1)))); %one for correct side estimate
end
for i = 1:length(thresh)
   I = find(score >= thresh(i));
   if ~isempty(I)
      count = count + 1;
      accuracy(count) = sum(pos_estimated(I) == pos(I))/length(I);

      J1 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 1);
      correct_side_dist1(count) = length(J1)/length(I);
      J2 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 2);
      correct_side_dist2(count) = length(J2)/length(I);
      J3 = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) == 3);
      correct_side_dist3(count) = length(J3)/length(I);
      Jh = find(correct_side(I) & abs(pos(I)- pos_estimated(I)) > 3);
      correct_side_disth(count) = length(Jh)/length(I);

      wrong_side(count) = sum(1-correct_side(I))/length(I);

      fraction(count) = length(I)/N;
   else
      count = count+1;
      accuracy(count) = NaN;
      correct_side_dist1(count) = NaN;
      correct_side_dist2(count) = NaN;
      correct_side_disth(count) = NaN;
      wrong_side(count) = NaN;
      fraction(count) = NaN;
   end
end

acc1 = accuracy + correct_side_dist1;
acc2 = accuracy + correct_side_dist1 + correct_side_dist2;
acc3 = accuracy + correct_side_dist1 + correct_side_dist2 + correct_side_dist3;
```

```
 %clf
 hold on

 plot(perc, acc3,'y','linewidth',2)
 plot(perc, acc2,'g','linewidth',2)
 plot(perc, acc1,'r','linewidth',2)
 plot(perc, accuracy,'b','linewidth',2)
 plot(perc, wrong_side,'k','linewidth',2)
 plot(perc, thresh,'c','linewidth',2)
 legend('dist \leq 3','dist \leq 2', 'dist \leq 1', 'precise', 'wrong side', 'threshold',2);
 xlabel('percentage');
 axis([0 100 0 1]);

 %keyboard
 %prepare result
 N = length(accuracy);
 res = [accuracy(N), acc1(N), acc2(N), acc3(N), 1-wrong_side(N), acc2(round(0.2*N))]
 return
function mfe = anti_inds_to_mfe(anti_inds)
% anti_inds holds for each nuc in the seq what is the index of
% the nuc across from it where the 0 means unpaired (this is returned by read_structure_withanti).
% returns mfe which is the structure in the format of rnafold, i.e. only base pairs:
% mfe is a 2 col matrix, the first being the bases on arm5 which are paired and the second
% their corresponding pairs
if(~iscell(anti_inds))
  mfe = get_mfe(anti_inds);
  return;
end
for i=1:length(anti_inds)
  mfe{i} = get_mfe(anti_inds{i});
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
function mfe = get_mfe(ai)
bps=0;
for i=1:length(ai)
  if(ai(i))
    if(i>ai(i))
      return
    end
    bps = bps+1;
    mfe(bps,1) = i;
    mfe(bps,2) = ai(i);
  end
end
%ktup, k, alpha
param_sets = [8,4,0.2;
  8,4,0.25;
  8,4,0.3;
  8,5,0.2;
```

```
  8,5,0.25;
  8,5,0.3;
  9,4,0.2;
  9,4,0.25;
  9,4,0.3];
fid = fopen('batch_results_proto4_A.txt','w');
params1;
maxd = 4;
set_name = model_params.trained_on;
fid = fopen(['zuker_draw_' set_name '.txt'],'r');
[palseq,anti_inds,bulges1,bulges2,endbulges,pal_id,energy,all_pal_ids] = ...
  read_structure_with_id_fid(fid,1000);
fclose(fid);
if(length(pal_id)~=length(all_pal_ids))
  error('in training data do not allow faulty seqs, take out of there');
end
mfes = anti_inds_to_mfe(anti_inds);
fname = ['mirseq_' set_name '.txt'];
[mirseq,mirlen] = read_seq_with_id(fname);
mirpos = locate_dicer(mirseq,palseq);
filename =['clust_proto_members_' num2str(maxd) '_' set_name '.txt'];
clust_num = load(filename);
if length(clust_num) ~= length(palseq)
  error('clust_num wrong size');
end
for i=1:size(params_sets,1)
  model_params.ktup = param_sets(i,1);
  model_params.k = param_sets(i,2);
  model_params.alpha = param_sets(i,3);
  [pos_est,edist_score,win_score] = mfold_cv_proto_members(mirseq,mirpos,mirlen,palseq,anti_inds,...
    bulges1,bulges2,endbulges,clust_num,model_params);
  score = edist_score; %!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  res = analyse_errors_perc(pos_est,score,mirpos,endbulges);
  fprintf(fid,'%d %d %4.2f   %5.3f %5.3f %5.3f %5.3f %5.3f\r\n', model_params.ktup, model_params.k, ...
    model_params.alpha,res(1),res(2),res(3),res(5),res(6));
end
fclose(fid);
function model = bayes_learn_win(seqs,anti_inds,bulges1,bulges2,endbulges,pos,mirlen,model)
%model_params is a struct.
ds_win_len = model.ds_win_len;
% mfes{i} holds the structure in the basepair notation
mfes = anti_inds_to_mfe(anti_inds);
% win_pos(i) is the position of the window corresponding to mir i
if(model.use_mirlen_in_learning_win)
  win_pos = get_win_pos_v1(mfes,anti_inds,pos,mirlen);
else
  win_pos = get_win_pos_v1(mfes,anti_inds,pos,ds_win_len*ones(size(pos)));
end
% for each seq hold the mirposition and all possible positions that are not mirpos
for i=1:length(pos)
```

```
  mirwin(i) = win_pos(i);
  ai = anti_inds{i};
  mfe = mfes{i};
  n_bps = size(mfes{i},1);
  tt = setdiff([1:n_bps],mirwin(i));
  nonmirwins_legal = [];
  for j=1:length(tt)
    wp=tt(j);
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    if((pos3_on_arm5>=model.min_win_len) & (length(ai)-pos5_on_arm3+1>=model.min_win_len))
      nonmirwins_legal = [nonmirwins_legal,wp];
    end
  end
  nonmirwin{i} = nonmirwins_legal;
end
[mean_loopdist,std_loopdist] = loopdist_bp_model_normal(win_pos,mfes);
model.mean_loopdist_bp = mean_loopdist;
model.std_loopdist_bp = std_loopdist;
[win_num_bps_mir_vals,win_num_bps_mir_ps] = num_bps_model_hist_list(mfes,anti_inds,model,mirwin);
[win_num_bps_nonmir_vals,win_num_bps_nonmir_ps] = num_bps_model_hist_list(mfes,anti_inds,model,nonmirwin);
model.win_num_bps_mir_vals = win_num_bps_mir_vals;
model.win_num_bps_mir_ps = win_num_bps_mir_ps;
model.win_num_bps_nonmir_vals = win_num_bps_nonmir_vals;
model.win_num_bps_nonmir_ps = win_num_bps_nonmir_ps;
[win_sym_mir_vals,win_sym_mir_ps] = win_sym_model_list(mfes,anti_inds,model,mirwin);
[win_sym_nonmir_vals,win_sym_nonmir_ps] = win_sym_model_list(mfes,anti_inds,model,nonmirwin);
model.win_sym_mir_vals = win_sym_mir_vals;
model.win_sym_mir_ps = win_sym_mir_ps;
model.win_sym_nonmir_vals = win_sym_nonmir_vals;
model.win_sym_nonmir_ps = win_sym_nonmir_ps;
[pb_arm5_mir,pb_arm3_mir,pb1_arm5_mir,pb1_arm3_mir,pb2_arm5_mir,pb2_arm3_mir]...
  = win_bulge_pos_model_list(mfes,bulges1,bulges2,model,mirwin);
[pb_arm5_nonmir,pb_arm3_nonmir,pb1_arm5_nonmir,pb1_arm3_nonmir,pb2_arm5_nonmir,pb2_arm3_nonmir]...
  = win_bulge_pos_model_list(mfes,bulges1,bulges2,model,nonmirwin);
model.win_bulge_posit_arm5_mir = pb_arm5_mir;
model.win_bulge_posit_arm3_mir = pb_arm3_mir;
model.win_bulge1_posit_arm5_mir = pb1_arm5_mir;
model.win_bulge1_posit_arm3_mir = pb1_arm3_mir;
model.win_bulge2_posit_arm5_mir = pb2_arm5_mir;
model.win_bulge2_posit_arm3_mir = pb2_arm3_mir;
model.win_bulge_posit_arm5_nonmir = pb_arm5_nonmir;
model.win_bulge_posit_arm3_nonmir = pb_arm3_nonmir;
model.win_bulge1_posit_arm5_nonmir = pb1_arm5_nonmir;
model.win_bulge1_posit_arm3_nonmir = pb1_arm3_nonmir;
model.win_bulge2_posit_arm5_nonmir = pb2_arm5_nonmir;
model.win_bulge2_posit_arm3_nonmir = pb2_arm3_nonmir;
[win_p_bp_arm5_mir,win_p_bp_arm3_mir] = ...
  win_base_pair_model_list(mfes,anti_inds,seqs,model,mirwin);
[win_p_bp_arm5_nonmir,win_p_bp_arm3_nonmir] =...
```

```
   win_base_pair_model_list(mfes,anti_inds,seqs,model,nonmirwin);
model.win_base_pair_arm5_mir = win_p_bp_arm5_mir;
model.win_base_pair_arm3_mir = win_p_bp_arm3_mir;
model.win_base_pair_arm5_nonmir = win_p_bp_arm5_nonmir;
model.win_base_pair_arm3_nonmir = win_p_bp_arm3_nonmir;
return
function [pos,combined_score, edist_score,win_score] = firstkpp_predict_combined(model,
seqs,anti_inds,bulges1,bulges2,endbulges);
% [pos,combined_score,edist_score,win_score] = firstkpp_predict_combined(model,
seqs,anti_inds,bulges1,bulges2,endbulges);
%
% predict best matching miRNA position by edit distance to the first k letters of known mirs
% from the best scoring positions, take the ones with best 2stage score
%
%model contains all learned model, that of bayesian predictor and all known mirs
%seqs is in int format. converted to nucleotide format inside firstkpp_predict1
%
% GD 21.10.03
disp('calculating...');
for i = 1:length(seqs)
   [posi, combined_scorei,edist_scorei, win_scorei] =
firstkpp_predict1(model,seqs{i},anti_inds{i},bulges1{i},bulges2{i},endbulges{i});
   pos(i) = posi;
   combined_score(i) = combined_scorei;
   edist_score(i) = edist_scorei;
   win_score(i) = win_scorei;
end
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [pos,combined_score, edist_score, win_score] = firstkpp_predict1(model,seqsi,
anti_indsi,bulges1i,bulges2i,endbulgesi);
%calculate the best matching position of dicer
min_win_len = model.min_win_len;
modelk = model.k;
ktup = model.ktup;
gamma = model.gamma;
lb = find(endbulgesi);
eb_begin = lb(1);
eb_end = lb(end);
%initialize variables with the largest possible distance
mean_k = ktup(ones(length(seqsi),1));
seqsi_nuc = int2nuc(seqsi);
%upper side
for i = 1:1:eb_begin-min_win_len
   p = seqsi_nuc(i:i+ktup-1);
   for j = 1:length(model.seqsd)
      d(j) = editD(p,model.seqsd{j});
   end
   % take also the mean of highest percentile
```

```
  [ds,I] = sort(d);
  mean_k(i) = mean(ds(1:modelk));
end
%lower side
for i = eb_end+1:1:length(seqsi)-min_win_len+1
  p = seqsi_nuc(i:i+ktup-1);
  for j = 1:length(model.seqsd)
    d(j) = editD(p,model.seqsd{j});
  end
  % take also the mean of highest ten percentile
  [ds,I] = sort(d);
  mean_k(i) = mean(ds(1:modelk));
end
%rewrite the last choosing of parameters
fk_score = 1 - model.beta*mean_k/ktup;
max_score = max(fk_score);
thrsh_score = (1-model.alpha)*max_score;
Ic = find(fk_score >= thrsh_score);
if(isempty(Ic))
  pos = nan;
  combined_score = nan;
  edist_score = nan;
  win_score = nan;
  return
end
% now compute two stage scores
twostg_score = win_score_2stagei(model,seqsi,anti_indsi,bulges1i,bulges2i,endbulgesi);
twostg_score = interpolate_nan(twostg_score,endbulgesi);
combined_score = gamma*fk_score(Ic) + (1-gamma)*twostg_score(Ic);
[max_combined, imx] = max(combined_score);
pos = Ic(imx);
combined_score = max_combined;
edist_score = fk_score(pos);
win_score = twostg_score(pos);
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function score_interp = interpolate_nan(score, endbulgesi);
% fill all NaNs which are surrounded by numeric values by interpolation
Ib = find(endbulgesi);
score(Ib) = 0;
A = find(isnan(score));
B = find(~isnan(score));
score_interp = zeros(size(score));
score_interp(B) = score(B);
score_interp(A) = interp1(B,score(B),A);
score_interp(find(isnan(score_interp))) = 0;
score_interp = score_interp';
return
function win_mirpos = get_win_pos_v1(mfes,anti_inds,mirpos,mirlen)
```

```
% function win_mirpos = get_win_pos(mfes,anti_inds,mirpos,mirlen)
% returns win_mirpos in index of basepair (from legs not loop).
% i.e. mfe(win_mirpos,1) is the nuc pos on the 5 arm
% for mir on arm3 returns the closest bp from its mirpos towards the legs
% for mir on arm5 returns the closest bp from its END (mirpos+mirlen-1) towards the legs
% also towards the legs
for i=1:length(mirpos)
  pos5 = mirpos(i);
  pos3 = pos5+mirlen(i)-1;
  mfe = mfes{i};
  arm5 = mfe(:,1);
  arm3 = mfe(:,2);
  eb_start = arm5(end)+1;
  eb_end = arm3(end)-1;
  eb_len = eb_end-eb_start+1;
  side5 = (pos5<eb_start);
  ai = anti_inds{i};
  is_paired = (ai~=0);
  if(side5)
    k=0;
    while(~is_paired(pos3-k))
      k=k+1;
    end
    win_mirpos(i) = find(arm5==(pos3-k));
  else
    k=0;
    while(~is_paired(pos5+k))
      k=k+1;
    end
    win_mirpos(i) = find(arm3==(pos5+k));
  end
  if(isempty(win_mirpos(i)))
    error('get_win_pos: fatal error. aborting.');
  end
end
function strseq = int2nuc(intseq, ncase)
%strseq = int2nuc(intseq, ncase)
%convert a sequence of '1 2 3 4' into  'A C T G'  or 'a c t g'
% ncase = uppercase | lowercase
if(isletter(intseq(1)))
  strseq = intseq;
  return;
end
if nargin == 1
  ncase = 'uppercase';
end
if strcmp(ncase,'uppercase')
  nucs = 'ACTG';
elseif strcmp(ncase,'lowercase')
  nucs = 'actg';
```

```
end
strseq = char(size(intseq));
for i = 1:length(intseq)
   strseq(i) = nucs(intseq(i));
end
return
function [yside, yprec2] = interpolate_prob_new(score, fitfile);
%[yside, yprec2] = interpolate_prob_new(score, fitfile);
% load the parameters for interpolation
load(fitfile);
%interpolate
yside = interp1(xs,ys,score,'linear');
yprec2 = interp1(xp2,yp2,score,'linear');
% extrapolate if necessary
if(min(xs)==xs(1)) % x is increasing
   yside(score<xs(1)) = ys(1);
   yprec2(score<xp2(1)) = yp2(1);
   yside(score>xs(end)) = ys(end);
   yprec2(score>xp2(end)) = yp2(end);
else % x is decreasing
   yside(score>xs(1)) = ys(1);
   yprec2(score>xp2(1)) = yp2(1);
   yside(score<xs(end)) = ys(end);
   yprec2(score<xp2(end)) = yp2(end);
end
returnfunction [mean_dist,std_dist] = loopdist_bp_model_normal(win_pos,mfes)
for i=1:length(win_pos)
   n_bps = size(mfes{i},1);
   loopdist(i) = n_bps - win_pos(i);
end
% cut off outliers
lp = prctile(loopdist,[2.5 97.5]);
l = find(loopdist >= lp(1) & loopdist <=lp(2));
mean_dist = mean(loopdist(l));
std_dist = std(loopdist(l));
%figure;hist(loopdist,[0:max(loopdist+1)]);title('loopdist training');function [pos_est,score,edist_score,win_score] =
mfold_cv_members(mirseq,mirpos,mirlen,palseq,anti_inds,bulges1,bulges2,...
   endbulges,clust_num,mfold,model_params);
%[pos_est,score,edist_score,win_score] =
mfold_cv_members(mirseq,mirpos,mirlen,palseq,anti_inds,bulges1,bulges2,...
% endbulges,clust_num,mfold,model_params);
n_all = length(palseq);
pos_est = zeros(0);
score =zeros(0);
model = model_params;
clust_list = unique(clust_num);
num_clusts = length(clust_list)
bins = round(0:num_clusts/mfold:num_clusts)
for m=1:mfold
   disp(['m= ' num2str(m)]);
```

```
  bs_clusts = clust_list([bins(m)+1: bins(m+1)]);
  bs = []; % test set
  for i=1:length(bs_clusts)
    this_clust = bs_clusts(i);
    bs = [bs;find(clust_num==this_clust)];
  end
  disp(['size test set: ' num2str(size(bs))]);
 bt = setdiff(1:n_all, bs);% train set

  disp('building model...');
 % learn model , and add all known mirs to it
 model = bayes_learn_win(palseq(bt),anti_inds(bt),bulges1(bt),bulges2(bt),endbulges(bt), ...
    mirpos(bt),mirlen(bt),model);
  clear seqsd_train;
  for i = 1:length(bt); seqsd_train{i} = mirseq{bt(i)}(1:model.ktup); end
  model.seqsd = transform_format(seqsd_train);

  disp('predicting...');
  [pos_est_m,score_m,edist_score_m,win_score_m] = firstkpp_predict_combined...
    (model, palseq(bs),anti_inds(bs),bulges1(bs),bulges2(bs),endbulges(bs));
  pos_est(bs) = pos_est_m;
  score(bs) = score_m;
  edist_score(bs) = edist_score_m;
  win_score(bs) = win_score_m;
end
returnfunction [pos_est,score,edist_score,win_score] =
mfold_cv_random(mirseq,mirpos,mirlen,palseq,anti_inds,bulges1,bulges2,...
  endbulges,mfold,randstate,model_params,permute);
%[pos_est,score,edist_score,win_score] =
mfold_cv_random(mirseq,mirpos,mirlen,palseq,anti_inds,bulges1,bulges2,...
% endbulges,mfold,randstate,model_params,permute);
if(~exist('permute'))
  permute = 1;
end
n_all = length(palseq);
bins = round(0:n_all/mfold:n_all)
bins_all = 1:n_all;
if permute
  rand('state',randstate);
 l = randperm(n_all);
  mirseq = mirseq(l);
  mirpos = mirpos(l);
  mirlen = mirlen(l);
  palseq = palseq(l);
  anti_inds = anti_inds(l);
  bulges1 = bulges1(l);
  bulges2 = bulges2(l);
  endbulges = endbulges(l);
end
pos_est = zeros(0);
```

```
edist_score =zeros(0);
win_score =zeros(0);
model = model_params;
m = 1;
while m <= mfold
  bs = [bins(m)+1: bins(m+1)];% test set
  bt = setdiff(bins_all , bs);% train set

  disp([ 'm = ' num2str(m)]);

  disp('building model...');
 % learn model , and add all known mirs to it
 model = bayes_learn_win(palseq(bt),anti_inds(bt),bulges1(bt),bulges2(bt),endbulges(bt), ...
     mirpos(bt),mirlen(bt),model);
  clear seqsd_train;
 for i = 1:length(bt); seqsd_train{i} = mirseq{bt(i)}(1:model.ktup); end
 model.seqsd = transform_format(seqsd_train);
  disp('predicting...');
  [pos_est_m,score_m,edist_score_m,win_score_m] = firstkpp_predict_combined...
     (model, palseq(bs),anti_inds(bs),bulges1(bs),bulges2(bs),endbulges(bs));
  pos_est(bs) = pos_est_m;
  score(bs) = score_m;
  edist_score(bs) = edist_score_m;
  win_score(bs) = win_score_m;

  m = m+1;
end
if permute
  % undo the permutation
  pos_est(I) = pos_est;
  score(I) = score;
  edist_score(I) = edist_score;
  win_score(I) = win_score;
end
returnfunction [intseq, fault_seq] = nuc2int(strseq);
%[intseq, fault_seq] = nuc2int(strseq)
%convert a sequence of 'A C T G' into a array of 1 2 3 4
if(~isletter(strseq(1)))
  intseq = strseq;
  fault_seq = 0;
  return;
end
intseq = zeros(size(strseq));
fault_seq = 0;
for i = 1:length(strseq)
  switch upper(strseq(i))
     case 'A' , intseq(i) = 1;
     case 'C' , intseq(i) = 2;
     case 'T' , intseq(i) = 3;
     case 'G' , intseq(i) = 4;
```

```
    otherwise , intseq = []; fault_seq = 1; break;
  end
end
function [num_bps_vals,num_bps_ps] = num_bps_model_hist_list(mfes,anti_inds,model,wps)
numseqs = length(wps);
if(numseqs~=length(mfes) | numseqs~=length(anti_inds))
  error('number of seqs differs from length(wps)');
end
% transform wps into cell if it is not so.
if(~iscell(wps))
  for i=1:numseqs
    tt{i} = wps(i);
  end
  wps = tt;
end
beta = 0.5;
win_len = model.ds_win_len;
num_bps = [];
for i=1:numseqs
  wp_list = wps{i};
  mfe = mfes{i};
  ai = anti_inds{i};
  is_paired = (ai~=0);
  for k=1:length(wp_list)
    wp = wp_list(k);
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
    pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
    numpaired5 = sum(is_paired(pos5_on_arm5:pos3_on_arm5));
    numpaired3 = sum(is_paired(pos5_on_arm3:pos3_on_arm3));
    num_bps = [num_bps,min(numpaired5,numpaired3)];
  end
end
num_bps_vals = 0:model.win_num_bins_num_bps-1;
n = hist(num_bps,num_bps_vals);
n = n+beta;
num_bps_ps = n/sum(n);
%figure;bar(num_bps_vals,num_bps_ps);title('numbps hist training');
model_params.trained_on = 'hmdcc440';
% see files with this extension for the training data itself
%specific firstk parameters
model_params.ktup = 8; % window size for edist part
model_params.k = 4;    % number of neareset neighbors in KNN
model_params.alpha = 0.25; %fraction best score that defines the region for ranking with 2stage
model_params.beta = 2;  %scaling parameter: score = 1-beta*mean_k/ktup;
model_params.gamma = 0.75; % the weight of the first (edist) score in combined score
% win params
model_params.min_win_len = 17; % single starnded min win len in nts.
model_params.ds_win_len = 22; % double starnded win len in nts.
```

```
model_params.use_mirlen_in_learning_win = 0; % if 1 uses mirlen else uses win_len in learning win
model_params.win_base_pair_states = 6; % this param is used only for win prediction.
model_params.win_bulge = 0; % for win prediction. which bulges to look at. 1/2 - bulges1/2, else total
model_params.win_num_bins_sym = model_params.ds_win_len;
model_params.win_num_bins_num_bps = model_params.ds_win_len;
model_params.win_use_loopdist = 1;
model_params.win_use_win_sym = 1;
model_params.win_use_pos_bulge = 1;
model_params.win_use_num_bps = 1;
model_params.win_use_base_pair = 0;
function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid(fid,seqtot)
% function [seqs,anti_inds,bulges_nonsym,bulges_sym,endbulges,pal_id,energy,all_pal_ids] =
read_structure_with_id_fid(fid,seqtot)
% same as read_structure_withanti_fid but reads file that have before the 4 line zuker draw
% a line giving the pal_id and a line giving the energy.
% all_pal_ids is all ids read from file, whether faulty or not
% new feature: checks that draw is not messed up and if it is gives faulty seq.
Mxplen = 250;  % maximal length of palindrom
counter = 0;
seq_no = 0;
seqs = cell(0);
bulges_nonsym= cell(0);
bulges_sym= cell(0);
endbulges = cell(0);
pal_id = zeros(0);
energy = zeros(0);
while ~feof(fid) & seq_no < seqtot
  this_pal_id = str2double(fgetl(fid));
  this_energy = str2double(fgetl(fid));
  structure = char(4,250);
  i = 0;
  line = fgetl(fid);
  if(isempty(line))
    line = 'emptyline';
    fault_seq_emptyline = 1;
  else
    fault_seq_emptyline = 0;
  end
  while(line(1)~='|')  % if emptyline this is always true so will go into loop
    i = i+1;
    structure(i,1:length(line)) = line;
    line = fgetl(fid);
    if(isempty(line))
      line = 'emptyline';
      fault_seq_emptyline = 1;
    end
  end
  if(i~=4)
    fault_seq_numlines = 1;
```

```
else
  fault_seq_numlines = 0;
end

fault_seq_struct = 1; % guilty until proven innocent
fault_seq_nuc = 1;
if(fault_seq_numlines == 0 & fault_seq_emptyline==0)
  [seqi, anti_indi, bulge1i, bulge2i, endbulgei,fault_seq_struct] = get_features(structure);
  if(fault_seq_struct==0)
    % this is the old bulge1 and bulge2, now need to correct that
    bulge_nonsymi=bulge1i;
    bulge_symi=bulge2i;
    for j = 1:length(seqi)
      if(bulge_nonsymi(j))
        if(bulge_symi(max(1,j-1))) % a neighbor has a bulgesym flag on
          bulge_symi(j) = 1;
          bulge_nonsymi(j) = 0;
        end
      end
    end
    for j = length(seqi):-1:1
      if(bulge_nonsymi(j))
        if(bulge_symi(min(j+1,length(seqi)))) % a neighbor has a bulgesym flag on
          bulge_symi(j) = 1;
          bulge_nonsymi(j) = 0;
        end
      end
    end
    [intseq, fault_seq_nuc] = nuc2int(seqi);
  end
end

if (fault_seq_struct == 0 & fault_seq_nuc == 0 & fault_seq_numlines == 0 & fault_seq_emptyline == 0)
    seq_no = seq_no + 1;
    seqs{seq_no} = intseq;
    anti_inds{seq_no} = anti_indi;
    bulges_nonsym{seq_no} = bulge_nonsymi;
    bulges_sym{seq_no} = bulge_symi;
    endbulges{seq_no} = endbulgei;
    pal_id(seq_no) = this_pal_id;
    energy(seq_no) = this_energy;
    counter = counter + 1;
    all_pal_ids(counter) = this_pal_id;
 else
    disp(['faulty seq on pal id  ' num2str(this_pal_id)])
    if(fault_seq_emptyline)
      disp(['reason is that there was an empty line in zuker']);
    elseif(fault_seq_numlines)
      disp(['reason is that there were not 4 lines in the draw']);
    elseif(fault_seq_struct)
```

```
        disp(['reason is that draw was messed has nuc in pair and bulge at the same time']);
      elseif(fault_seq_nuc)
        disp(['reason is that there was an illegal letter in the seq']);
      end
      counter = counter + 1;
      all_pal_ids(counter) = this_pal_id;
    end
end
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [seq, anti_ind, bulge1, bulge2, endbulge, fault_seq] = get_features(structure)
% get sequence as well as bulge structure
fault_seq = 0;
%upper half (5' side)
bulge_row = 1; % the row of bulge letters
bulge_row_opposite = 4;
uphalf = structure(1:2,:);
[j,k] = find(isletter(uphalf));
max_col = max(k);
tmpmat = zeros(2,max_col);
count = 0;
for col =1: max_col
  fl = find(isletter(uphalf(:,col)));
  if (length(fl)>1);
    fault_seq = 1;
    seq=nan;anti_ind=nan;bulge1=nan;bulge2=nan;endbulge=nan;
    return;
  end;
  if ~isempty(fl)
    count = count + 1;
    seq(count) = uphalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(1,col) = 0;
    else
      tmpmat(1,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
  end
end
% endbulge is coded on the upper half
% go backwards form 3' side to 5' side
endbulge = zeros(size(bulge1));
```

```
lwhalf = structure(3:4,:);
pos = length(bulge1);
while bulge1(pos) == 1
  endbulge(pos) = 1;
  bulge1(pos) = 0;
  pos = pos - 1;
end
%lower half
bulge_row = 2; % 4 th line on structure is 2 line on lower half
bulge_row_opposite = 1;
[j,k] = find(isletter(lwhalf));
max_col = max(k);
for col =max_col:-1:1
  fl = find(isletter(lwhalf(:,col)));
  if ~isempty(fl)
    count = count + 1;
    seq(count) = lwhalf(fl,col);
    bulge = (fl == bulge_row);
    if(bulge)
      tmpmat(2,col) = 0;
    else
      tmpmat(2,col) = count;
    end
    bulge1(count) = 0;
    bulge2(count) = 0;
    if bulge & isletter(structure(bulge_row_opposite,col))
      bulge2(count) = 1;
    elseif bulge & ~isletter(structure(bulge_row_opposite,col))
      bulge1(count) = 1;
    end
    endbulge(count) = 0;
  end
end
anti_ind = zeros(size(bulge1));
for col=1:max_col
  if(tmpmat(1,col))
    anti_ind(tmpmat(1,col)) = tmpmat(2,col);
    anti_ind(tmpmat(2,col)) = tmpmat(1,col);
  end
end
return
function run_firstkpp(infile, outfile)
%run_firstkpp(infile, outfile)
model_filename = 'model_hmdcc440_params1.mat';
fitfile = 'fitfile_hmdcc440_params1_mfold5_proto5.mat';
fidin = fopen(infile,'r');
fidout = fopen(outfile,'a');
seqstot = 1000; %number of sequences to classify each loop
load(model_filename);
while ~feof(fidin)
```

```
  disp('reading structure...');
  [palseq,anti_inds,bulges1,bulges2,endbulges,pal_id,energy,all_pal_ids] = ...
    read_structure_with_id_fid(fidin,seqstot);
  mfes = anti_inds_to_mfe(anti_inds);
  [pos_est,score,edist_score,win_score] = ...
    firstkpp_predict_combined(model,palseq,anti_inds,bulges1,bulges2,endbulges);
  [yside, yprec2] = interpolate_prob_new(score, fitfile);
  res = [pal_id; pos_est; score; yprec2; edist_score; win_score];
  fprintf(fidout, '%d %d %g %g %g %g\r\n', res);
end
fclose(fidin);
fclose(fidout);
param_file='params1'; params1;
model = model_params;
model.param_file = param_file;
set_name = model_params.trained_on;
fid = fopen(['zuker_draw_' set_name '.txt'],'r');
[palseq,anti_inds,bulges1,bulges2,endbulges,pal_id,energy,all_pal_ids] = ...
  read_structure_with_id_fid(fid,1000);
fclose(fid);
if(length(pal_id)~=length(all_pal_ids))
  error('in training data do not allow faulty seqs, take out of there');
end
mfes = anti_inds_to_mfe(anti_inds);
fname = ['mirseq_' set_name '.txt'];
[mirseq,mirlen] = read_seq_with_id(fname);
mirpos = locate_dicer(mirseq,palseq);
extension = [set_name '_params1'];
maxd = 5;
mfold = 5;
extension_proto = [set_name '_params1_mfold5_proto5'];
randstate = 1;
extension_random = [set_name '_params1_mfold5_randstate1'];
disp('building model from all data and saving it....')
% learn model , and add all known mirs to it
model = bayes_learn_win(palseq,anti_inds,bulges1,bulges2,endbulges,mirpos,mirlen,model);
% take the first ktup nucleotides of every miR
for i = 1:length(mirseq); mirseq{i} = mirseq{i}(1:model.ktup); end
model.seqsd = transform_format(mirseq);
eval(['save model_' extension '.mat model']);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% random mfold %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
if(1)
disp('doing random mfold cv....')
[pos_est,score,edist_score,win_score] = mfold_cv_random(mirseq,mirpos,mirlen,palseq,anti_inds,...
  bulges1,bulges2,endbulges,mfold,randstate,model_params,1);
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est,score,mirpos,endbulges);
```

```
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] = analyse_errors_bins2(pos_est,score,mirpos,endbulges,num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
eval(['print -djpeg ' extension_random '.jpeg']);
eval(['save fitfile_' extension_random '.mat xs ys xp2 yp2']);
figure;
fid = fopen(['info_and_criteria_' extension_random '.txt'],'w');
thresh_vec = [0:0.01:1];
clf;[thresh,acc2,captures] = analyse_errors_thresh_B(pos_est,score,mirpos,endbulges,thresh_vec);
grid
legend('off')
fprintf(fid,'%%thresh\tacc2\tcaptures\r\n');
for i=1:length(thresh)
  fprintf(fid,'%1.4f\t%1.4f\t%d\r\n',thresh(i),acc2(i),captures(i));
end
fclose(fid);
%save mfold results for each pal individually
fitfile = ['fitfile_' extension_random];
[yside, yprec2] = interpolate_prob_new(score, fitfile);
fid = fopen(['all_pal_res_' extension_random '.txt'],'w');
fprintf(fid,'%%pal_id\treal_mirpos\tfirstkpp_pos\tfirstkpp_score\typrec2\tfirstkpp_edist_score\tfirstk++_win_score\r\n');
fprintf(fid,'%%-----------------------------------------------------------------\r\n');
palres = [pal_id; mirpos; pos_est; score; yprec2; edist_score; win_score];
fprintf(fid, '%d %d %d %g %g %g %g\r\n', palres);
fclose(fid);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%% memebers mfold %%%%%%
if(1)
disp('doing proto cv....')
filename =['clust_proto_members_' num2str(maxd) '_' set_name '.txt'];
clust_num = load(filename);
if length(clust_num) ~= length(palseq)
  error('clust_num wrong size');
end
[pos_est,score,edist_score,win_score] = mfold_cv_members(mirseq,mirpos,mirlen,palseq,anti_inds,...
  bulges1,bulges2,endbulges,clust_num,mfold,model_params);
figure
subplot(2,1,1)
res = analyse_errors_perc(pos_est,score,mirpos,endbulges);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
subplot(2,1,2)
```

```
if(~exist('num_bins'))
  num_bins = 6;
end
[xs,ys,xp2,yp2] = analyse_errors_bins2(pos_est,score,mirpos,endbulges,num_bins);
a=axis; a(3)=0; a(4)=1; axis(a); grid;
legend('off')
eval(['print -djpeg ' extension_proto '.jpeg']);
eval(['save fitfile_' extension_proto '.mat xs ys xp2 yp2']);
figure;
fid = fopen(['info_and_criteria_' extension_proto '.txt'],'w');
thresh_vec = [0:0.01:1];
clf;[thresh,acc2,captures] = analyse_errors_thresh_B(pos_est,score,mirpos,endbulges,thresh_vec);
grid
legend('off')
fprintf(fid,'%%thresh\tacc2\tcaptures\r\n');
for i=1:length(thresh)
  fprintf(fid,'%1.4f\t%1.4f\t%d\r\n',thresh(i),acc2(i),captures(i));
end
fclose(fid);
%save mfold results for each pal individually
fitfile = ['fitfile_' extension_proto];
[yside, yprec2] = interpolate_prob_new(score, fitfile);
fid = fopen(['all_pal_res_' extension_proto '.txt'],'w');
fprintf(fid,'%%pal_id\treal_mirpos\tfirstkpp_pos\tfirstkpp_score\typrec2\tfirstkpp_edist_score\tfirstk++_win_score\r\n');
fprintf(fid,'%%----------\r\n');
palres = [pal_id; mirpos; pos_est; score; yprec2; edist_score; win_score];
fprintf(fid, '%d %d %d %g %g %g %g\r\n', palres);
fclose(fid);
end
function seqs = transform_format(seqs,format);
%seqs = transform_format(seqs,format);
% format is either 'int' or 'nuc'
%if format not given, toggle format from int<-> nuc
% note that assume all seqs are in same format initially
if(nargin==1)
  if all(isletter(seqs{1}))
    format = 'int';
  else
    format = 'nuc';
  end
end

if(strcmp(format,'nuc'))
 for i = 1:length(seqs)
    seqs{i} = int2nuc(seqs{i});
  end
elseif(strcmp(format,'int'))
  for i = 1:length(seqs)
    seqs{i} = nuc2int(seqs{i});
  end
```

```
else
  error('transform_format: format (if given) must be int or nuc');
end
return

function [p_bp_arm5,p_bp_arm3] = win_base_pair_model_list(mfes,anti_inds,seqs,model,wps)
numseqs = length(wps);
if(numseqs~=length(mfes) | numseqs~=length(anti_inds) | numseqs~=length(seqs))
  error('number of seqs differs from length(wps)');
end
% transform wps into cell if it is not so.
if(~iscell(wps))
  for i=1:numseqs
    tt{i} = wps(i);
  end
  wps = tt;
end
win_len = model.ds_win_len;
base_pair_states = model.win_base_pair_states;
c_bp_arm5 = zeros(1,base_pair_states);
c_bp_arm3 = zeros(1,base_pair_states);
seqsbp = nuc2bp(seqs,anti_inds,base_pair_states);
for i = 1:numseqs
  wp_list = wps{i};
  mfe = mfes{i};
  ai = anti_inds{i};
  is_paired = (ai~=0);
  for k=1:length(wp_list)
    wp = wp_list(k);
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
    pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
    for j = 1:base_pair_states
      c_bp_arm5(j) = c_bp_arm5(j)+sum(seqsbp{i}(pos5_on_arm5:pos3_on_arm5) == j);
      c_bp_arm3(j) = c_bp_arm3(j)+sum(seqsbp{i}(pos5_on_arm3:pos3_on_arm3) == j);
    end
  end
end
p_bp_arm5 = c_bp_arm5/sum(c_bp_arm5);
p_bp_arm3 = c_bp_arm3/sum(c_bp_arm3);
function [pb_arm5,pb_arm3,pb1_arm5,pb1_arm3,pb2_arm5,pb2_arm3] = ...
  win_bulge_pos_model_list(mfes,bulges1,bulges2,model,wps)
% on both sides of window from loop end of window
% pb1 - for bulges1 pb2 - for bulges2 pb - for total
win_len = model.ds_win_len;
numseqs = length(wps);
if(numseqs~=length(mfes) | numseqs~=length(bulges1) | numseqs~=length(bulges2))
  error('number of seqs differs from length(wps)');
end
```

```
% transform wps into cell if it is not so.
if(~iscell(wps))
  for i=1:numseqs
    tt{i} = wps(i);
  end
  wps = tt;
end
for i=1:numseqs
  wp_list = wps{i};
  mfe = mfes{i};
  bulges{i} = bulges1{i}+bulges2{i};
  inds5_i = cell(0);
  inds3_i = cell(0);
  for k=1:length(wp_list)
    wp = wp_list(k);
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
    pos3_on_arm3 = min(length(bulges{i}),pos5_on_arm3+win_len-1);
    inds5_i{k} = pos3_on_arm5:-1:pos5_on_arm5; % always start from loop side
    inds3_i{k} = pos5_on_arm3:pos3_on_arm3;
  end
  inds5{i} = inds5_i;
  inds3{i} = inds3_i;
end
pb_arm5 = bulge_positional_list(model,bulges,inds5);
pb_arm3 = bulge_positional_list(model,bulges,inds3);
pb1_arm5 = bulge_positional_list(model,bulges1,inds5);
pb1_arm3 = bulge_positional_list(model,bulges1,inds3);
pb2_arm5 = bulge_positional_list(model,bulges2,inds5);
pb2_arm3 = bulge_positional_list(model,bulges2,inds3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function p = bulge_positional_list(model,bulges,inds)
win_len = model.ds_win_len;
c = zeros(win_len,2);
p = zeros(win_len,1);
for i = 1:length(bulges)
  bulgesi = bulges{i};
  for k = 1:length(inds{i})
    this_inds = inds{i}{k};
    for j=1:length(this_inds)
      this_ind = this_inds(j);
      c(j,1) = c(j,1) + bulgesi(this_ind);
      c(j,2) = c(j,2) + (1-bulgesi(this_ind));
    end
  end
end
for j = 1:win_len
  p(j) = c(j,1)/sum(c(j,:));
```

```
end
function pos_scorei = win_score_2stagei(model,seqsi,anti_indsi,bulges1i,bulges2i,endbulgesi)
%function pos_scorei = win_score_2stagei(model,seqsi,anti_indsi,bulges1i,bulges2i,endbulgesi);
% pos_score is a vector having the length of the ith pal. pos_scorei(j) is the
% score of the window which gives that position of the pal. The entry is
% NULL if no window produces that pos5 or if it is on a loop. Note that each
% double stranded window gives two pos5, one on each arm, and they both have the same
% score - that of the ds_win.
mfesi = anti_inds_to_mfe(anti_indsi);
pos_scorei = get_pos_scores(model,seqsi,mfesi,anti_indsi,bulges1i,bulges2i,endbulgesi);
return
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function pos_scores = get_pos_scores(model,seqsi,mfei,ai,bulges1i,bulges2i, endbulgesi);
pos_scores = nan * ones(1,length(seqsi)); % initially all nan
p_mir = ones(1,size(mfei,1));
p_nonmir = ones(1,size(mfei,1));
wp_scores = nan * ones(1,size(mfei,1)); % in base pairs
if(model.win_use_loopdist)
 p_loopdist = loopdist_bp_prob_normal(model,mfei);
   p_mir = p_mir.*p_loopdist;
   p_nonmir = p_nonmir.*(1-p_loopdist);
end
if(model.win_use_num_bps)
 [p_num_bps_mir,p_num_bps_nonmir] = num_bps_prob_hist(model,mfei,ai);
   p_mir = p_mir.*p_num_bps_mir;
   p_nonmir = p_nonmir.*p_num_bps_nonmir;
end
if(model.win_use_win_sym)
 [p_win_sym_mir,p_win_sym_nonmir] = win_sym_prob(model,mfei,ai);
   p_mir = p_mir.*p_win_sym_mir;
   p_nonmir = p_nonmir.*p_win_sym_nonmir;
end
if(model.win_use_pos_bulge)
 [p_pos_bulge_mir,p_pos_bulge_nonmir] = win_bulges_pos_prob(model,mfei,bulges1i,bulges2i,0);
   p_mir = p_mir.*p_pos_bulge_mir;
   p_nonmir = p_nonmir.*p_pos_bulge_nonmir;
end
if(model.win_use_base_pair)
 [p_base_pair_mir,p_base_pair_nonmir] = win_base_pair_prob(model,mfei,ai,seqsi);
   p_mir = p_mir.*p_base_pair_mir;
   p_nonmir = p_nonmir.*p_base_pair_nonmir;
end
I = find((p_mir + p_nonmir) > 0);
wp_scores(I) = p_mir(I)./(p_mir(I)+p_nonmir(I));
% now transfer each of the win scores to the positions scores
for wp=1:length(wp_scores)
   s = wp_scores(wp);
   if(~isnan(s))
      pos3_on_arm5 = mfei(wp,1);
```

```
    pos5_on_arm3 = mfei(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-model.ds_win_len+1);
    pos_scores(pos5_on_arm3) = s;
    pos_scores(pos5_on_arm5) = s;
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function p_loopdist = loopdist_bp_prob_normal(model,mfe);
n_bps = size(mfe,1);
wp = 1:n_bps;
zloopdist = ((n_bps - wp) - model.mean_loopdist_bp)/model.std_loopdist_bp;
p_loopdist = exp(-0.5*zloopdist.^2);
p_loopdist = p_loopdist/sum(p_loopdist);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [p_num_bps_mir,p_num_bps_nonmir] = num_bps_prob_hist(model,mfe,ai);
win_len = model.ds_win_len;
n_bps = size(mfe,1);
p_num_bps_mir = zeros(1,n_bps);
p_num_bps_nonmir = zeros(1,n_bps);
is_paired = (ai~=0);
for wp = 1:n_bps
  pos3_on_arm5 = mfe(wp,1);
  pos5_on_arm3 = mfe(wp,2);
  pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
  pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
  win5inds = pos5_on_arm5:pos3_on_arm5;
  win3inds = pos5_on_arm3:pos3_on_arm3;
  if((length(win5inds)>=model.min_win_len) & (length(win3inds)>=model.min_win_len))
    numpaired5 = sum(is_paired(win5inds));
    numpaired3 = sum(is_paired(win3inds));
    num_bps_i = min(numpaired5,numpaired3);
    % mir
    tt = find(model.win_num_bps_mir_vals == num_bps_i);
    if(tt)
      p_num_bps_mir_i = model.win_num_bps_mir_ps(tt);
    else
      p_num_bps_mir_i = 0;
    end
    p_num_bps_mir_i = p_num_bps_mir_i*(win_len/mean(length(win5inds),length(win3inds)));
    p_num_bps_mir(wp) = p_num_bps_mir_i;
    % nonmir
    tt = find(model.win_num_bps_nonmir_vals == num_bps_i);
    if(tt)
      p_num_bps_nonmir_i = model.win_num_bps_nonmir_ps(tt);
    else
      p_num_bps_nonmir_i = 0;
    end
    p_num_bps_nonmir_i = p_num_bps_nonmir_i*(win_len/mean(length(win5inds),length(win3inds)));
```

```
      p_num_bps_nonmir(wp) = p_num_bps_nonmir_i;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [p_win_sym_mir,p_win_sym_nonmir] = win_sym_prob(model,mfe,ai);
win_len = model.ds_win_len;
n_bps = size(mfe,1);
p_win_sym_mir = zeros(1,n_bps);
p_win_sym_nonmir = zeros(1,n_bps);
is_paired = (ai~=0);
for wp = 1:n_bps
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
    pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
    win5inds = pos5_on_arm5:pos3_on_arm5;
    win3inds = pos5_on_arm3:pos3_on_arm3;
    if((length(win5inds)>=model.min_win_len) & (length(win3inds)>=model.min_win_len))
      numunpaired5 = sum(~is_paired(win5inds));
      numunpaired3 = sum(~is_paired(win3inds));
      win_sym_i = abs(numunpaired5-numunpaired3);
      % mir
      tt = find(model.win_sym_mir_vals == win_sym_i);
      if(tt)
        p_win_sym_mir_i = model.win_sym_mir_ps(tt);
      else
        p_win_sym_mir_i = 0;
      end
      p_win_sym_mir_i = p_win_sym_mir_i*sqrt(win_len/mean(length(win5inds),length(win3inds)));
      p_win_sym_mir(wp) = p_win_sym_mir_i;
      % nonmir
      tt = find(model.win_sym_nonmir_vals == win_sym_i);
      if(tt)
        p_win_sym_nonmir_i = model.win_sym_nonmir_ps(tt);
      else
        p_win_sym_nonmir_i = 0;
      end
      p_win_sym_nonmir_i = p_win_sym_nonmir_i*sqrt(win_len/mean(length(win5inds),length(win3inds)));
      p_win_sym_nonmir(wp) = p_win_sym_nonmir_i;
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [p_pos_bulge_mir,p_pos_bulge_nonmir] = win_bulges_pos_prob(model,mfe,bulges1i,bulges2i,use_avg);
bulge_flag = model.win_bulge;
win_len = model.ds_win_len;
n_bps = size(mfe,1);
p_pos_bulge_mir = zeros(1,n_bps);
p_pos_bulge_nonmir = zeros(1,n_bps);
```

```
pb_arm5_mir = model.win_bulge_posit_arm5_mir;
pb_arm3_mir = model.win_bulge_posit_arm3_mir;
pb1_arm5_mir = model.win_bulge1_posit_arm5_mir;
pb1_arm3_mir = model.win_bulge1_posit_arm3_mir;
pb2_arm5_mir = model.win_bulge2_posit_arm5_mir;
pb2_arm3_mir = model.win_bulge2_posit_arm3_mir;
pb_arm5_nonmir = model.win_bulge_posit_arm5_nonmir;
pb_arm3_nonmir = model.win_bulge_posit_arm3_nonmir;
pb1_arm5_nonmir = model.win_bulge1_posit_arm5_nonmir;
pb1_arm3_nonmir = model.win_bulge1_posit_arm3_nonmir;
pb2_arm5_nonmir = model.win_bulge2_posit_arm5_nonmir;
pb2_arm3_nonmir = model.win_bulge2_posit_arm3_nonmir;
if(use_avg)
  pb_mir = 0.5*(pb_arm5_mir+pb_arm3_mir);
  pb_arm5_mir = pb_mir;
  pb_arm3_mir = pb_mir;
  pb1_mir = 0.5*(pb1_arm5_mir+pb1_arm3_mir);
  pb1_arm5_mir = pb1_mir;
  pb1_arm3_mir = pb1_mir;
  pb2_mir = 0.5*(pb2_arm5_mir+pb2_arm3_mir);
  pb2_arm5_mir = pb2_mir;
  pb2_arm3_mir = pb2_mir;
  pb_nonmir = 0.5*(pb_arm5_nonmir+pb_arm3_nonmir);
  pb_arm5_nonmir = pb_nonmir;
  pb_arm3_nonmir = pb_nonmir;
  pb1_nonmir = 0.5*(pb1_arm5_nonmir+pb1_arm3_nonmir);
  pb1_arm5_nonmir = pb1_nonmir;
  pb1_arm3_nonmir = pb1_nonmir;
  pb2_nonmir = 0.5*(pb2_arm5_nonmir+pb2_arm3_nonmir);
  pb2_arm5_nonmir = pb2_nonmir;
  pb2_arm3_nonmir = pb2_nonmir;
end
if(bulge_flag == 1)
  pb_arm5_mir = pb1_arm5_mir;
  pb_arm3_mir = pb1_arm3_mir;
  pb_arm5_nonmir = pb1_arm5_nonmir;
  pb_arm3_nonmir = pb1_arm3_nonmir;
  bulgesi = bulges1i;
elseif(bulge_flag == 2)
  pb_arm5_mir = pb2_arm5_mir;
  pb_arm3_mir = pb2_arm3_mir;
  pb_arm5_nonmir = pb2_arm5_nonmir;
  pb_arm3_nonmir = pb2_arm3_nonmir;
  bulgesi = bulges2i;
else
  % just use the total pb.
  bulgesi = bulges1i+bulges2i;
end
for wp = 1:n_bps
  pos3_on_arm5 = mfe(wp,1);
```

```
  pos5_on_arm3 = mfe(wp,2);
  pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
  pos3_on_arm3 = min(length(bulgesi),pos5_on_arm3+win_len-1);
  win5 = bulgesi(pos3_on_arm5:-1:pos5_on_arm5); % always start from loop side
  win3 = bulgesi(pos5_on_arm3:pos3_on_arm3);
  win5_len_actual = length(win5);
  win3_len_actual = length(win3);
  if((length(win5)>=model.min_win_len) & (length(win3)>=model.min_win_len))
    J0 = find(win5 == 0);
    J1 = find(win5);
    p_bulges5_mir_i = prod(pb_arm5_mir(J1)) * prod(1-pb_arm5_mir(J0));
    p_bulges5_mir_i = p_bulges5_mir_i^(win_len/win5_len_actual);
    p_bulges5_nonmir_i = prod(pb_arm5_nonmir(J1)) * prod(1-pb_arm5_nonmir(J0));
    p_bulges5_nonmir_i = p_bulges5_nonmir_i^(win_len/win5_len_actual);
    J0 = find(win3 == 0);
    J1 = find(win3);
    p_bulges3_mir_i = prod(pb_arm3_mir(J1)) * prod(1-pb_arm3_mir(J0));
    p_bulges3_mir_i = p_bulges3_mir_i^(win_len/win3_len_actual);
    p_bulges3_nonmir_i = prod(pb_arm3_nonmir(J1)) * prod(1-pb_arm3_nonmir(J0));
    p_bulges3_nonmir_i = p_bulges3_nonmir_i^(win_len/win3_len_actual);

    p_pos_bulge_mir(wp) = sqrt(p_bulges5_mir_i*p_bulges3_mir_i);
    p_pos_bulge_nonmir(wp) = sqrt(p_bulges5_nonmir_i*p_bulges3_nonmir_i);
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [p_base_pair_mir,p_base_pair_nonmir] = win_base_pair_prob(model,mfe,ai,seq);
win_len = model.ds_win_len;
base_pair_states = model.win_base_pair_states;
p_bp_arm5_mir = model.win_base_pair_arm5_mir;
p_bp_arm3_mir = model.win_base_pair_arm3_mir;
p_bp_arm5_nonmir = model.win_base_pair_arm5_nonmir;
p_bp_arm3_nonmir = model.win_base_pair_arm3_nonmir;
n_bps = size(mfe,1);
p_base_pair_mir = zeros(1,n_bps);
p_base_pair_nonmir = zeros(1,n_bps);
t1{1} = seq;
t2{1} = ai;
t3 = nuc2bp(t1,t2,base_pair_states);
seqbp = t3{1};
for wp = 1:n_bps
  pos3_on_arm5 = mfe(wp,1);
  pos5_on_arm3 = mfe(wp,2);
  pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
  pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
  win5inds = (pos5_on_arm5:pos3_on_arm5);
  win3inds = (pos5_on_arm3:pos3_on_arm3);
  if((length(win5inds)>=model.min_win_len) & (length(win3inds)>=model.min_win_len))
    % mir
```

```
    p5_mir_i = 1;
    p3_mir_i = 1;
    for j = 1:base_pair_states
      p5_mir_i = p5_mir_i * p_bp_arm5_mir(j)^sum(seqbp(win5inds) == j);
      p3_mir_i = p3_mir_i * p_bp_arm3_mir(j)^sum(seqbp(win3inds) == j);
    end
    p5_mir_i = p5_mir_i.^(win_len/length(win5inds));
    p3_mir_i = p3_mir_i.^(win_len/length(win3inds));
    p_base_pair_mir(wp) =  sqrt(p5_mir_i*p3_mir_i);
    % nonmir
    p5_nonmir_i = 1;
    p3_nonmir_i = 1;
    for j = 1:base_pair_states
      p5_nonmir_i = p5_nonmir_i * p_bp_arm5_nonmir(j)^sum(seqbp(win5inds) == j);
      p3_nonmir_i = p3_nonmir_i * p_bp_arm3_nonmir(j)^sum(seqbp(win3inds) == j);
    end
    p5_nonmir_i = p5_nonmir_i.^(win_len/length(win5inds));
    p3_nonmir_i = p3_nonmir_i.^(win_len/length(win3inds));
    p_base_pair_nonmir(wp) =  sqrt(p5_nonmir_i*p3_nonmir_i);
  end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  [win_sym_vals,win_sym_ps] = win_sym_model_list(mfes,anti_inds,model,wps)
numseqs = length(wps);
if(numseqs~=length(mfes) | numseqs~=length(anti_inds))
  error('number of seqs differs from length(wps)');
end
% transform wps into cell if it is not so.
if(~iscell(wps))
  for i=1:numseqs
    tt{i} = wps(i);
  end
  wps = tt;
end
beta = 0.5;
win_len = model.ds_win_len;
win_sym = [];
for i=1:numseqs
  wp_list = wps{i};
  mfe = mfes{i};
  ai = anti_inds{i};
  is_paired = (ai~=0);
  for k=1:length(wp_list)
    wp = wp_list(k);
    pos3_on_arm5 = mfe(wp,1);
    pos5_on_arm3 = mfe(wp,2);
    pos5_on_arm5 = max(1,pos3_on_arm5-win_len+1);
    pos3_on_arm3 = min(length(ai),pos5_on_arm3+win_len-1);
    numunpaired5 = sum(~is_paired(pos5_on_arm5:pos3_on_arm5));
```

```
      numunpaired3 = sum(~is_paired(pos5_on_arm3:pos3_on_arm3));
      win_sym = [win_sym,abs(numunpaired5-numunpaired3)];
    end
  end
  win_sym_vals = 0:model.win_num_bins_sym-1;
  n = hist(win_sym,win_sym_vals);
  n = n+beta;
  win_sym_ps = n/sum(n);
  %figure;bar(win_sym_vals,win_sym_ps);title('win sym training');
```